Building eCommerce Applications

Articles for Developers by Developers from DevZone

For more great articles and tutorials check out DevZone:
https://www.x.com/developers/paypal/devzone

For the latest news in the Commerce space,
check out Commerce Weekly on O'Reilly Radar:
http://oreil.ly/commerceweekly

O'REILLY®
oreilly.com

Building eCommerce Applications

Developers from DevZone

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Building eCommerce Applications
by Developers from DevZone

Copyright © 2011 O'Reilly Media. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Mary Treseler
Production Editor: Kristen Borg
Proofreader: Rachel Monaghan
Cover Designer: Mark Paglietti
Interior Designer: David Futato
Illustrator: Robert Romano

October 2011: First Edition.

Revision History for the First Edition:
See *http://oreilly.com/catalog/errata.csp?isbn=9781449316891* for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Building eCommerce Applications* and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-31689-1

[LSI]

1317663918

Table of Contents

Introduction ..... **ix**

1. Creating a Mobile Enabled Shopping Cart with HTML5, jQuery, and TaffyDB, Part 1 ..... **1**
Step 1: Set Up the Tutorial Directory on Your Web Server 1
Step 2: Provide the Entire Product Catalog in JSON Format from the eCommerce System 2
Step 3: Set Up the Catalog HTML Page 4
Step 4: Load the Product Catalog in JSON Format from the eCommerce System 5
Step 5: Display the Store View 6
Step 6: Display the Category View 8
Step 7: Display the Product View 9
Step 8: Style the Links and Buttons 11
That's It 12

2. Creating a Mobile Enabled Shopping Cart with HTML5, jQuery, and TaffyDB, Part 2 ..... **15**
Step 1: Bind the Add to Cart Button to the addToCart() Function 20
Step 2: Initialize the HTML5 Cart Database 20
Step 3: Store the Product Information in the cartItems Database 22
Step 4: Add the HTML for the Cart View and Bind the View Cart Button to the demo.viewCart() Function 25
Step 5: Display the Cart 26
Step 6: Format the Cart View 30
Step 7: Stringify the Cart to Send to the Server 30
Step 8: Clear the Cart Contents 31
That's It 31

3. Analyzing Open Data for Fun (and Profit?), Part 1 33
Getting the World Bank Data 34
Loading the Data into MySQL 34
Downloading Google Visualization Tools 36
Configuring a Visualization Web App with Tomcat 37
Implementing and Querying a SQL Data Source 38
Next Steps... 40

4. Analyzing Open Data for Fun (and Profit?), Part 2 41
Implementing Table and Line Chart Visualizations 41
Creating an Event-Driven Line Chart Application 45
Next Steps 50
Until Next Time 50

5. Analyzing Open Data for Fun (and Profit?), Part 3 51
PayPal X: Your Global Payment Platform 51
Dollars and Sense 52
Adaptive Payments Configuration 54
V-Rupees for Graphs 55
Generating a Pay Key with the AdaptivePayments API 57
Kickstart the Embedded Payments Flow 60
Handling Payment Responses 64
Wrapping Up 66

6. Build an MLM Engine with Neo4j and MassPay, Part 1 67
What Is Multi-Level Marketing? 67
MLMs Are Graphs 68
The Design and Implementation of an MLM Engine 69
Traversing the Graph to Compute Commissions 76
Wrapping Things Up 80

7. Build an MLM Engine with Neo4j and MassPay, Part 2 81
Introduction 81
Defining the Class 82
Authentication and API Resolution 83
Making the Call 85
Putting It All Together 87

8. Getting Started with PayPal on Django 89
Introduction 89
Requirements 89

Paypal Accounts 89
Django 90
Sample Application 90
Running the Sample Application 90
Configuration 90
Database 90
Web Server 91
A Basic Payment Workflow 91
The Model 91
URL Mappings 91
Step 1—List Content Available for Purchase 92
Step 2—Attempt to Access Content 92
Step 3—Purchase Content 93
Step 4—Verify Payment 94
Step 5—Successful Purchase 96
Further Enhancements 97
Conclusion 97
Further Information and Other Resources 97

9. Accelerate Your Development Using the Apigee API Console 99
What Is the Apigee PayPal API Console? 99
APIs Available to You in the Console 100
Using the Console 101
Linking Calls Together 103
Other Apigee Consoles You Might Want to Check Out 104
Accelerate Your Own Development 105
About the Author 105

10. Interview with iConcessionStand 107
About the Author 115

11. Magento, Part 1: How to Install Magento 117
Comparing the Editions 117
Server Requirements 118
Installing Magento, Step 1: Download, Extract, and Upload 118
Installing Magento, Step 2: Set Up Permissions 119
Installing Magento, Step 3: Creating the Database 119
Installing Magento, Step 4: Run the Setup Wizard 120
Installing Magento, Step 5: Verify the Installation 121
Next Steps 121
About the Author 122

12. Magento, Part 2: How to Set Up Your Magento Store 123
Creating Categories 123
Adding a Product in your Magento Store 125
Wrapping Up 134
About the Author 134

13. Magento, Part 3: Accepting PayPal Payments 137
How to Accept Payments with PayPal in Magento 138
Wrapping Up 144
About the Author 145

14. Magento, Part 4: Integrating Magento and WordPress Using the FishPig Extension 147
Integrating Magento and WordPress with FishPig 148
Wrapping Up 158
About the Author 159

15. The Convergence of Local, Social, and Mobile, Part 1: Local 161
Breaking Silos 162
Defining Local 162
The Evolution of Local 163
Phase 1: Pre–personal transportation and technology 163
Phase 2: Introduction of personal transportation 164
Phase 3: The information "superhighway" 165
Share Your Thoughts 166
About the Author 167

16. The Convergence of Local, Social, and Mobile, Part 2: Social 169
Briefly Defining Social 170
The Evolution of Social 170
Phase 1: "Can I borrow your goat?" 170
Phase 2: Pen pals, trade, and travel 171
Phase 3: The rise of instant communication 172
A Final Note on Social Media 173
Share Your Thoughts 173
About the Author 173

17. The Convergence of Local, Social, and Mobile, Part 3: Mobile 175
The Evolution of Mobile 176
Phase 1: Can you hear me now? 176
Phase 2: The "G" wars 177

Phase 3: The phone gets a brain and learns to surf 178
Stay Tuned 180
Share Your Thoughts 180
About the Author 180

18. The Convergence of Local, Social, and Mobile, Part 4: The Convergence .. 181
First, A Look at the Data 182
What Does This Mean for the Future of Local, Social, and Mobile? 183
Share Your Thoughts 185
About the Author 185

19. To Catch a (Quantum) Thief 187
Data Security in the Quantum Age 187
The Downside 188
Bad News 189
Hacking Accelerated 189
Game Changer 190
5 to 10 190

20. 10 Best Practices for Employing QR Codes 193
Size Matters (the Bigger, the Better) 194
Short Is Sweet 194
Location Matters 194
Know Why You're Using It 195
Make It Count 195
A Little Goes a Long Way 195
Tell People How to Use It 196
Provide an Alternative 196
Respect the Border 196
Bring the Wow 196
About the Author 196

21. Integrating Payments into WordPress, Part 1: Features and Getting Started . 197
What Is WordPress? 198
WordPress Documentation 199
Installing WordPress 200
First Steps After Installation 201
Coming Next Time 202
About the Author 202

22. Integrating Payments into WordPress, Part 2: PayPal Plugins 203
Extending WordPress with Plugins 203
Plugin Recommendations 204
PayPal Plugins 206
Plugin Learnings and Where They'll Take Us 210
About the Author 211

23. Integrating Payments into WordPress, Part 3: Build Your Own PayPal Plugin . 213
A Note About PHP 213
Understanding "The Loop" 214
The Anatomy of a WordPress Plugin 214
PayPal Functionality in Our Plugin 218
Deploying and Testing the Plugin 218
Things to Learn from This WordPress Series 220
About the Author 220

24. Crowd Marketing 221
Viral Isn't a Plan—It Is a Consequence 221
Making a Good Fire 222
What Happened? 222
Life Ahead Is eBay Without Having to Ship Stock from the Attic 223
Let Them Buy Wholesale and Distribute 223
Fans, Friends, Launchpads, and New Markets 224
About the Author 224

25. Bitcoin: Money Without Governments 225
What Is Bitcoin? 225
Acquiring and Storing Bitcoins 227
Using and Accepting Bitcoins 230
Threats and Opportunities 230
About the Author 232

Introduction

This collection of articles and blog entries is representative of the full spectrum of commerce-related content we've published on PayPal's Developer Network over the past year. You will find tutorials and quick reference pieces for developers. With the creation of x.commerce, we've expanded our coverage to address the needs of eBay and Magento developers, and you can expect to see more content focused on helping both the developer and merchant communities in the coming year.

Our team has covered a wide variety of topics: building mobile shopping carts, QR codes, working with various PayPal APIs, including how to integrate PayPal with other technologies such as WordPress. Three main themes have emerged in the commerce world today: Mobile, Social, and Local. Expect to see more coverage of these in the coming months.

If you haven't already visited, we hope you will join at DevZone: *https://www.x.com/developers/paypal/devzone*

Feel free to drop me a line if there are topics you would like us to cover or if you would like to write for us.

—*Mary Treseler*

Editorial Strategist, O'Reilly Media Inc.

mary@oreilly.com

CHAPTER 1

Creating a Mobile Enabled Shopping Cart with HTML5, jQuery, and TaffyDB, Part 1

Humberto Roa

The changing mobile landscape, and newer web technologies, means that working with mobile commerce requires a new approach to an old problem. This tutorial will take you through the steps to create a simple mobile commerce site that provides an interactive shopping experience over low-bandwidth connections. Part one of this tutorial will walk you through using PHP5, jQuery, and Taffy DB to create an interactive product catalog. Part two of this tutorial will walk you through using an HTML5 database to store a shopping cart and show you how to submit the shopping cart to a server at the time of checkout.

If you'd like to follow along, you can download the files for the project template here. The final source files that you'll have created by the end of this tutorial are here.

Step 1: Set Up the Tutorial Directory on Your Web Server

You will be able to complete this tutorial using a standard web server that has PHP5 installed. To get started, create a new directory on your web server named *tutorial*. Inside the tutorial directory, create two directories: *css* and *js*. Under the *js* directory, create a directory named *lib*.

When you are done, the directory structure should look like this:

tutorial/css/

tutorial/js/

tutorial/js/lib/

Next, download the following source libraries:

- *html5resetcss* - *http://code.google.com/p/html5resetcss/downloads/list*
- *jQuery 1.4.2* - *http://jquery.com*
- *Taffy DB 1.7.2* - *http://taffydb.com*
- *JSON2* - *http://www.json.org*

Place the file *html5resetcss.css* in the *tutorial/css/* directory, and place the *jQuery*, *Taffy DB*, and *JSON* javascript files in the *tutorial/js/lib/* directory.

Finally, create the following four empty files.

tutorial/catalog.php

tutorial/index.html

tutorial/js/site.js

tutorial/css/site.css

We'll will be working with these four files in this tutorial.

Step 2: Provide the Entire Product Catalog in JSON Format from the eCommerce System

In order to minimize http requests between the mobile device and the server while a customer is browsing the product catalog, we're going to download the entire catalog from the ecommerce system when the catalog is viewed in a browser. Note that if you are working with a very large catalog, you may want to download categories or subcategories as needed instead of the full catalog.

We'll start with defining the category and product information with arrays. In practice, these arrays will contain the results of database queries from your PHP5-based ecommerce system. For this tutorial, open the file *catalog.php*, and add the following category and product information:

```
<?php

// simulates result of db query for categories
$categories = array();
$categories[] = array(id => 1, parent_id => 0, name => 'root');
$categories[] = array(id => 2, parent_id => 1,
    name => 'Compact Discs');
```

```
$categories[] = array(id => 3, parent_id => 1,
    name => 'Concert Souvenirs');

// simulates result of db query for products
$products = array();
$products[] = array(id => 1, category_id => 2, sku => 'CD001',
    price=>15.00, name => 'CD: Greatest Hits');
$products[] = array(id => 2, category_id => 2, sku => 'CD002',
    price=>15.00, name => 'CD: Unplugged');
$products[] = array(id => 3, category_id => 2, sku => 'CD003',
    price=>15.00, name => 'CD: World Tour');
$products[] = array(id => 4, category_id => 3, sku => 'PD001',
    price=>10.00, name => 'Souvenir Pin');
$products[] = array(id => 5, category_id => 3, sku => 'PD002',
    price=>10.00, name => 'Mug');
$products[] = array(id => 6, category_id => 3, sku => 'PD003',
    price=>20.00, name => 'Hat');
$products[] = array(id => 7, category_id => 3, sku => 'PD004',
    price=>12.00, name => 'Summer Tour Poster');
$products[] = array(id => 8, category_id => 3, sku => 'PD005',
    price=>5.00,  name => 'Concert Program');
?>
```

Notice that the categories have a `parent_id field`, which contains the id of the parent category, and a `category_id field`, which contains the id of the product's category. These fields will be used later in this tutorial to find child categories and products for a parent category id.

Next, we'll convert the category and product information to JSON objects and arrays using the `json_encode` function that is provided by PHP5. Append the following code to the end of *catalog.php*:

```
// create the response
$response = array(categories => $categories, products => $products);

// display the json encoded response
echo json_encode($response);
```

That's it. When *catalog.php* is viewed in a browser, the category and product information will be provided as JSON objects and arrays. You can try it out by opening *catalog.php* in a browser.

I find it helpful to validate the JSON output at *http://www.jslint.com* to ensure the conversion is working correctly. Copy the output from *catalog.php*, paste it into the JSLint form, and submit. If you get the result message: "JSON: good.", you can move on to "Step 3: Set Up the Catalog HTML Page" on page 4.

Step 3: Set Up the Catalog HTML Page

For this tutorial, customers will be able to browse our mobile commerce site using three views:

1. The store view, which displays a list of product categories in the store.
2. The category view, which displays a list of products in a category.
3. The product view, which displays the details of a product.

To increase the responsiveness of our mobile commerce site on a mobile device, we'll be using a single HTML page to present all three views, and we'll use JavaScript and CSS to change the visibility of views and to update the view content as customers browse the site. We'll start by defining the basic structure for our mobile commerce site. Open the file named *index.html*, and add the following code:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
        <title>Concert Catalog</title>

        <!-- iOS -->
        <meta name="apple-mobile-web-app-capable" content="yes">
        <meta name="apple-mobile-web-app-status-bar-style" content="black">
        <meta name = "viewport" content = "width = device-width,
initial-scale = 1, user-scalable = no">

        <style type="text/css" media="screen">
            <!--
            @import url(css/html5reset-1.4.1.css);
            @import url(css/site.css);
            -->
        </style>

        <script type="text/javascript" src="js/lib/jquery-1.4.2.min.js">
</script>
        <script type="text/javascript" src="js/lib/taffy-min.js"></script>
        <script type="text/javascript" src="js/lib/json2.js"></script>
        <script type="text/javascript" src="js/site.js"></script>

    </head>
    <body></body>
</html>
```

Next, we'll add containers to *index.html* to display the store, category, and product views. Add the following code inside of the body element:

```
<div id="content">
    <div id="store"></div>
```

```
    <div id="category"></div>
    <div id="product"></div>
</div>
```

Step 4: Load the Product Catalog in JSON Format from the eCommerce System

Each time the mobile commerce site is viewed, the product catalog will be retrieved from the server in JSON format.

We'll start by creating a function in the file *site.js* to get the catalog JSON data using the jQuery ajax function. Open the file *site.js*, and add the following code:

```
var demo = {};
demo.categories = null;
demo.products = null;

demo.viewStore = function()
{

}

demo.loadCatalog = function()
{
    $.ajax( {
        url : 'catalog.php',

        success : function(result) {
            // parse json result
            var catalog = JSON.parse(result);

            // init Taffy DB collections
            demo.categories = new TAFFY(catalog.categories);
            demo.products = new TAFFY(catalog.products);

            // view the store
            demo.viewStore();
        },

        error: function (request, status, error) {
                    alert(request.responseText);
            }
    });

}

$(function(){
    demo.loadCatalog();
});
```

When the document is ready, the function `demo.loadCatalog` will be called, which will make an ajax request to get the output of the file *catalog.php*. If the request is successful, the response text will be passed to the success function, which will parse the text into a JavaScript object, and assign the object to a local variable named `catalog`.

Next, the success function will create Taffy DB collections for the categories and products using the catalog content. The collections will be assigned to the variables: `demo.categories` and `demo.products`.

Finally, the success function will call `demo.viewStore`, which will update the contents of the store view and display the store view to the customer.

If you are not familiar with Taffy DB, it is a great tool that simplifies working with collections of JavaScript objects. We'll use it in the next sections to populate the data in the store, category, and product views.

Step 5: Display the Store View

Now that the catalog data has been loaded into Taffy DB collections, we can start working to displaying the catalog content to the customer. We'll start with the store view by adding HTML elements to the div with the id store to hold the store name and the list of store categories. Replace the div with the id store with the following code:

```
<div id="store">
<nav>
    <h1 class="name">Our Band's Concert Store</h1>
</nav>
    <div class="category-list"></div>
</div>
```

Next we'll update the `demo.viewStore` JavaScript function in order to get the category data from the Taffy DB collections, create the category list HTML content, and bind JavaScipt to the category links. Replace the function `demo.viewStore` with the following code:

```
demo.setCurrentView = function(viewId)
{
    // remove the class: current from all the views
    $('#content > *').removeClass('current');

    // add the class: current to the view with the id viewId
    $(viewId).addClass('current');
}

demo.viewStore = function()
{
    demo.setCurrentView('#store');
```

```
    // get the data from TaffyDB
    var categories = demo.categories.get({parent_id:1});

    // create the html for the navigation links
    var html = '<ul>';
    for(i in categories)
    {
        html += '<li>';
        html += '<a href="#" id="' + categories[i]['id'] + '">' +
categories[i]['name'] + '</a>';
        html += '</li>';
    }
    html += '</ul>';

    // add the html to the view
    $('#store .category-list').empty().append(html);

    // bind the navigation links
    $('#store .category-list a').click(function(e){
        e.preventDefault();

        var id = parseInt($(this).attr('id'), 10);
        demo.viewCategory(id);
    });
}
```

When the function `demo.viewStore` is called, the function will first set the store view to be the current view by calling the function `demo.setCurrentView`. This will add the class "current" to the div with the id store.

The function will get the categories from the Taffy DB category collection where the `parent_id` of the category is equal to 1. This will return an array of category objects, which is then used to create links to the categories.

Next, the onclick function of the category links is changed to open the category view using the function `demo.viewCategory`.

Finally, we'll update *site.css* to hide all the views, except the current view. Open *site.css*, and add the following code:

```
#content {
    width: 320px;
    min-height: 480px;
    background-color: lightyellow;
}

#content > div {
    display: none;
    width: 320px;
    min-height: 480px;
    padding: 10px;
}
```

```
#content > .current {
    display: block;
}
```

Step 6: Display the Category View

Similar to store view, we'll start by adding HTML elements to the div with the id category to hold the category name and the list of category products. Replace the div with the id category with the following code:

```
<div id="category">
       <nav>
           <button class="back">Back</button>
        <h1 class="name"></h1>
       </nav>
       <div class="product-list"></div>
</div>
```

Next we'll add the `demo.viewCategory` JavaAcript function in order to get the category and product data from the Taffy DB collections, create the product list HTML content, and bind JavaScipt to the product links. Open the file *site.js*, and add the following code after the function `demo.viewStore`:

```
demo.viewCategory = function(categoryId)
{
    demo.setCurrentView('#category');

    // get the data from TaffyDB
    var category = demo.categories.get({id:categoryId}).pop();
    var products = demo.products.get({category_id: categoryId});

    // create the html for the navigation links
    var html = '<ul>';
    for(i in products)
    {
        html += '<li>';
        html += '<a href="#" id="' + products[i]['id'] + '">' + products[i]
['name'] + '</a>';
        html += '</li>';
    }
    html += '</ul>';

    // add the thml to the view
    $('#category .product-list').empty().append(html);

    // update the category name
       $('#category .name').text(category.name);

    // bind the navigation links
    $('#category .product-list a').click(function(e){
        e.preventDefault();
```

```
            var id = $(this).attr('id');
            demo.viewProduct(id);
        });

        // bind the back button
        $('#category .back').click(function(e){
            e.preventDefault();
            demo.viewStore();
        });
    }
```

When the function `demo.viewCategory` is called, the function will first set the category view to be the current view by calling the function `demo.setCurrentView`. This will add the class "current" to the div with the id category.

Next, the function will get the category from the Taffy DB category collection where the id of the category is equal to `categoryId`, and the function will get the products from the Taffy DB product collection where the `category_id` of the product is equal to `categoryId`. This will return an array of product objects, which is then used to create links to the products.

Finally, the onclick function of the product links is changed to open the product view using the function `demo.viewProduct`. Also, the onclick function of the back button is changed to open the store view using the function `demo.viewStore`.

You should now be able to open the store view and browse to a category view.

Step 7: Display the Product View

Similar to the store and category views, we'll start by adding HTML elements to the div with the id product to hold the product details. Replace the div with id product with the following code:

```
<div id="product">
    <div class="message"></div>
        <nav>
            <button class="back">Back</button>
            <h1>Product Details</h1>
        </nav>
        <div class="product-details">
            <dl>
                <dt>Name</dt><dd class="name"></dd>
                <dt>SKU</dt><dd class="sku"></dd>
                <dt>Price</dt><dd class="price"></dd>
                <dt>Quantity</dt><dd class="quantity"><input type="text" />
</dd>
            </dl>
            <button class="add-to-cart">+ Add to Cart</button>
```

```
        </div>
</div>
```

Next we'll add the `demo.viewProduct` JavaScript function in order to get the product data from the Taffy DB product collection, create the product detail HTML content, and bind JavaScipt to the links. Open the file *site.js*, and add the following code after the function `demo.viewCategory`:

```
demo.formatStringAsCurrency = function(str)
{
    var price = parseFloat(str, 10);
    return '$'+price.toFixed(2);
};

demo.viewProduct = function(productId)
{
    demo.setCurrentView('#product');

    // get the data from TaffyDB
    var product = demo.products.get({id:productId}).pop();
    var category = demo.categories.get({id:product.category_id}).pop();

    // update the product details
    $('#product .name').text(product.name);
    $('#product .sku').text(product.sku);
    $('#product .price').text(demo.formatStringAsCurrency(product.price));

    // bind the back button
    $('#product .back').click(function(e){
        e.preventDefault();
        demo.viewCategory(product.category_id);
    });
}
```

When the function `demo.viewProduct` is called, the function will first set the product view to be the current view by calling the function `demo.setCurrentView`. This will add the class "current" to the div with the id product.

Next, the function will get the product from the Taffy DB product collection where the id of the product is equal to `productId`, and the function will get the category from the Taffy DB category collection where the id of the category is equal to the `category_id` of the product. This will return an array of category and product objects. Notice that the array function `pop` is used to get the category and product objects from the arrays.

The function populates the HTML elements with the product attributes; notice that the function `demo.formatStringAsCurrency` is used to format the price value. The onclick function of the back button is changed to open the category view using `category_id` attribute of the product.

Last, we'll update *site.css* to format the display of the product details. Open *site.css*, and add the following code:

```
dl {
    width: 300px;
}

dt, dd {
    display: inline-block;
    line-height: 30px;
    border-top: 1px dashed silver;
}

dt:first-child, dd:nth-child(2) {
    border-top: none;
}

dt {
    width: 25%;
    font-weight: bold;
}

dd {
    width: 75%;
}
```

You should now be able to browse the catalog by clicking the category and product links. You can move back up to the previous view by clicking the back button at the top of the view.

Step 8: Style the Links and Buttons

For the final step, we'll update *site.css* to format the display of the links and buttons. Open the file *site.css*, and add the following code:

```
body {
    font-family: Helvetica, sans-serif;
}

a {
    -webkit-tap-highlight-color: rgba(150, 150, 150,.5);
}

h1 {
    margin-bottom: 5px;
}

button {
    height: 30px;
}
```

```
nav {
    width: 300px;
    height: 44px;
    line-height: 44px;
    margin-bottom: 5px;
}

    nav h1 {
        display: inline-block;
    }

    nav .back {
        margin-right: 20px;
    }

ul {
    width: 300px;
}

ul a {
    height: 44px;
    line-height: 44px;
    padding-left: 10px;
    border: 1px solid silver;
    border-top: none;
    color: #222;
    text-decoration: none;
    display: block;
    background-color: white;
}

ul li:first-child a {
    border-top: 1px solid silver;
}

#product .add-to-cart {
    margin-top: 20px;
    width: 300px;
}

.message {
    display: none;
}
```

That's It

That's it! You should now have a simple working mobile commerce site that allows you to interactively browse a product catalog using a single web page. In part two of this tutorial, we'll walk through using an HTML5 database to store a shopping cart and how to submit the shopping cart to a server at the time of checkout.

As a reminder, the final source files for this part of the tutorial are here.

CHAPTER 2

Creating a Mobile Enabled Shopping Cart with HTML5, jQuery, and TaffyDB, Part 2

Humberto Roa

The changing mobile landscape and emerging web technologies dictate that working with mobile commerce requires a new approach to an old problem. In Part 2 of this tutorial (see Chapter 1 for Part 1) for creating a simple mobile commerce site that provides an interactive shopping experience over low-bandwidth connections, I will walk through using an HTML5 database to store a shopping cart. Once that's wrapped up, I'll show you how to submit the shopping cart to a server at the time of checkout.

You will be able to complete this tutorial using a standard web browser that has support for HTML5 databases and a text editor. If you'd like to follow along, you can download the files for the project template here. The final source files that you'll have created by the end of this tutorial are here.

Screenshots of the completed app are shown below, and you can interact with a live demo of the final project here.

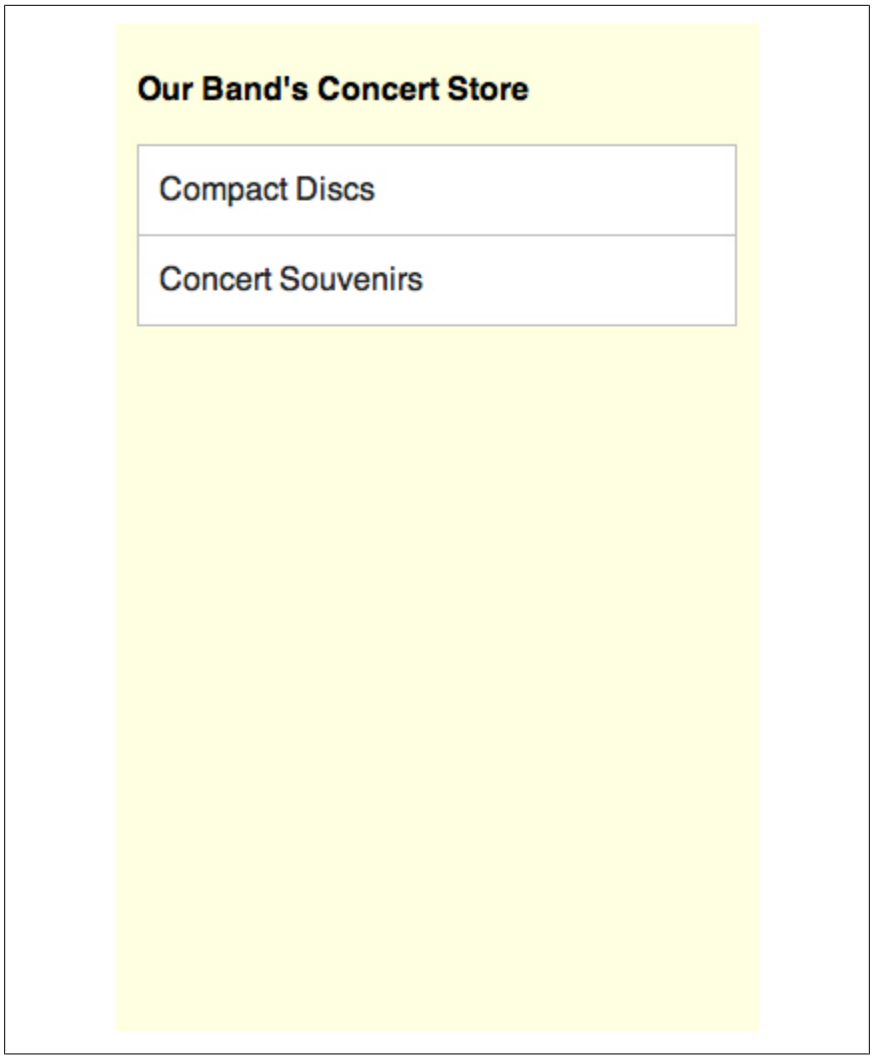


Figure 2-1. Screenshot of the final app; Store View

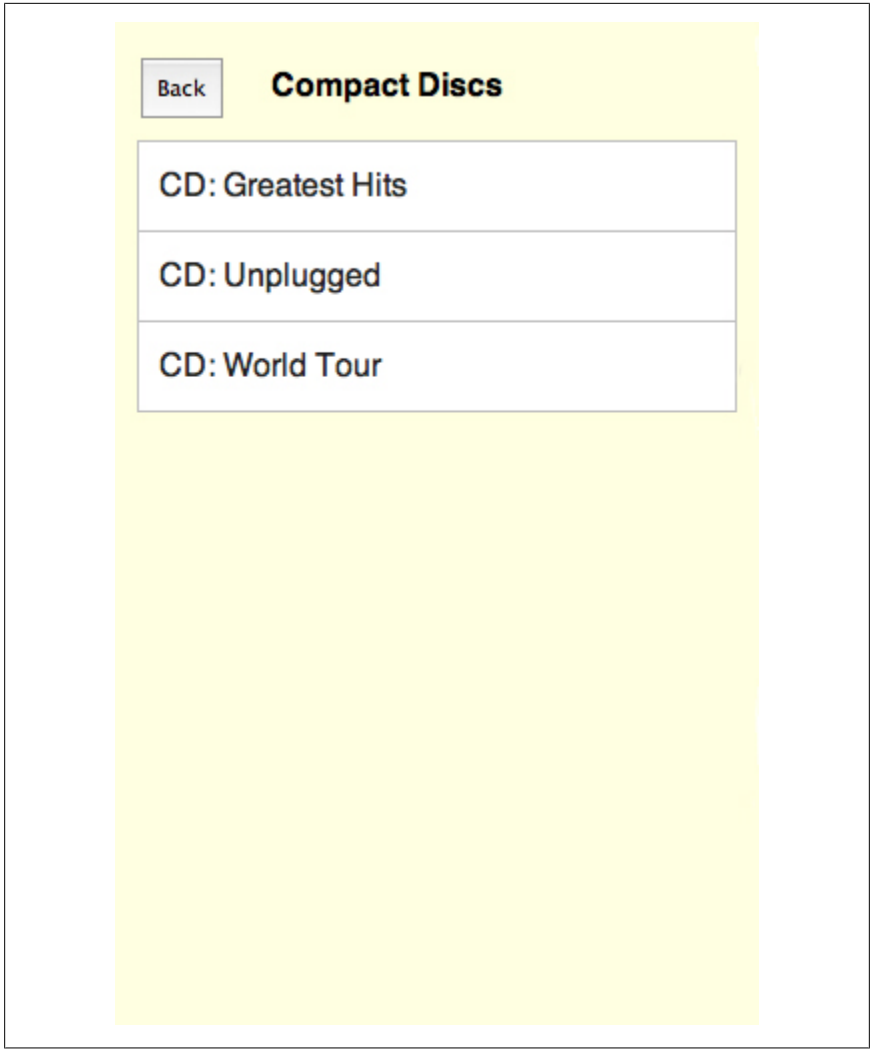


Figure 2-2. Screenshot of the final app; Category View

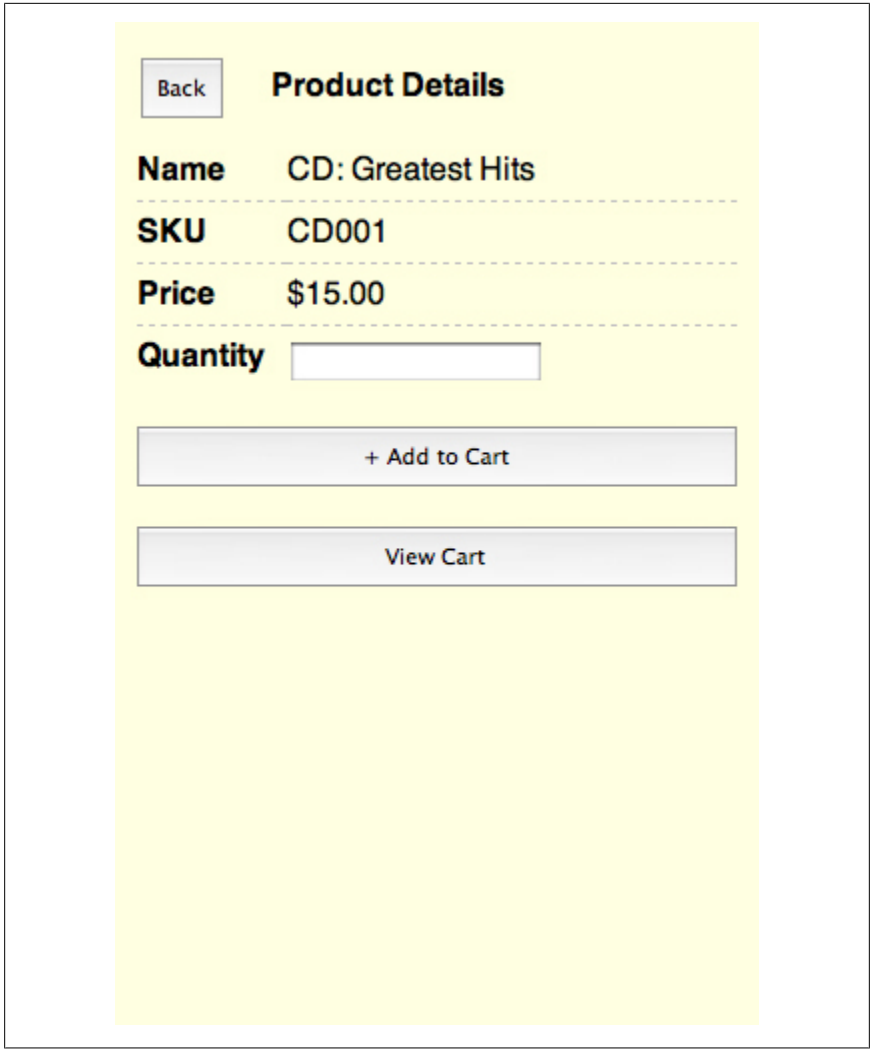


Figure 2-3. Screenshot of the final app; Product View

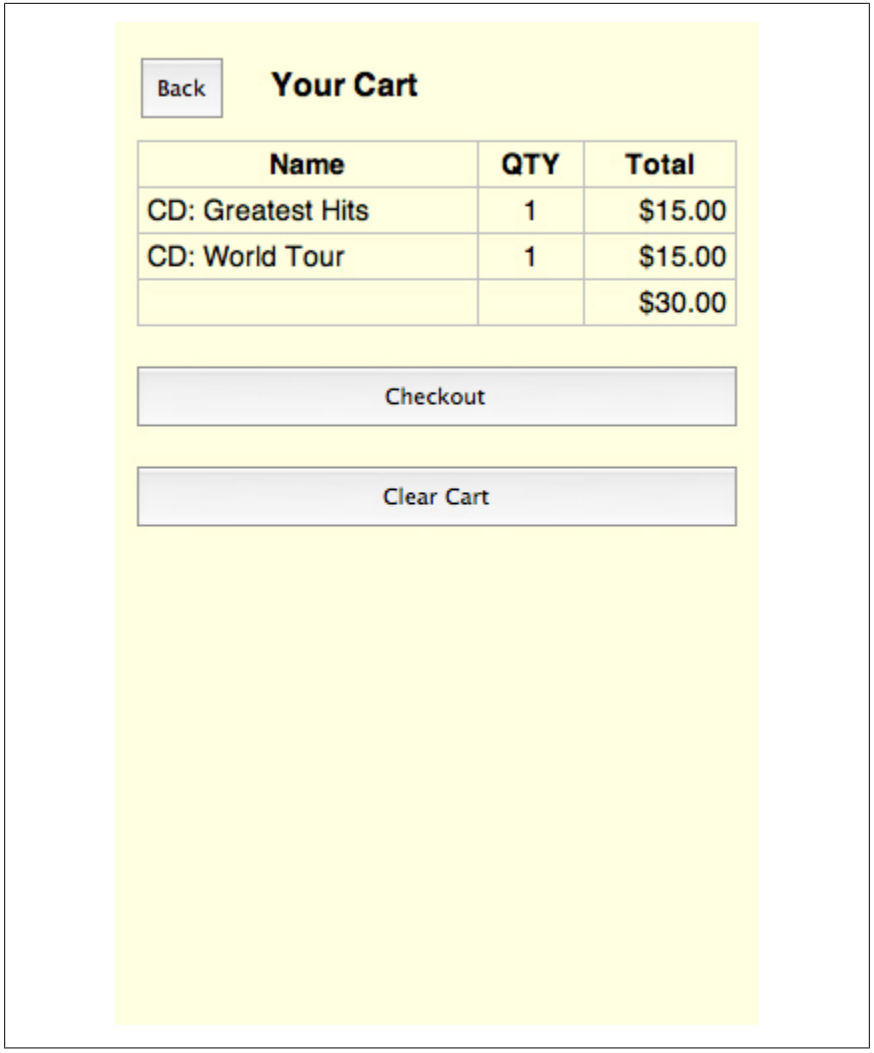


Figure 2-4. Screenshot of the final app; Cart View

Step 1: Bind the Add to Cart Button to the addToCart() Function

We'll start with updating the `demo.viewProduct()` function, created in Part 1, to bind the add to cart button to the `demo.addToCart()` function. Open the file *site.js*, and scroll down to the `demo.viewProduct()` function. Add the following code to the end of the function:

```
// bind the add to cart button
$('#product .add-to-cart').unbind('click').click(function(e){
    // add the item to the cart
    demo.addToCart(productId);
});
```

This code will call the `demo.addToCart()` function when the add to cart button is clicked. Notice that the jQuery `unbind()` function is called to unbind click event functions from the button before binding the new function to the button.

Next, add the following code after `demo.viewProduct` to create a place holder for the `addToCart()` function.

```
demo.addToCart = function(productId)
{
    // get the product from Taffy DB
    var product = demo.products.get({id:productId}).pop();

    console.log(product);
};
```

This will get the product from Taffy DB using the productId, and the product object will be displayed in the Developer Tools console each time the button is clicked. You can test the button to verify it is working correctly.

Step 2: Initialize the HTML5 Cart Database

The HTML database for the cart needs to be initialized before products can be added to the cart. In the file *site.js*, add the following code after the `demo.viewProduct()` function:

```
demo.db = null;

demo.initCartDB = function(){
    // database attributes
    var shortName = 'Cart';
    var version = '1.0';
    var displayName = 'Cart';
    var maxSize = 65536;

    // open the database
    this.db = openDatabase(shortName, version, displayName, maxSize);
```

```
    // create the cartItems table
    this.db.transaction(
        function(transaction) {
            transaction.executeSql(
                // SQL statement to create the cart table
                'CREATE TABLE IF NOT EXISTS cartItems ' +
                ' (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,' +
                ' item_id INTEGER, name TEXT, price REAL,
                '  quantity INTEGER);'
            );
        }
    );
};
```

First, the variable `db` is added to the demo object to hold a reference to the database after it is initialized.

```
demo.db = null;
```

Next, the `demo.initCartDB()` function is defined, and the database attributes that are required to open our cart database are set in local variables.

```
demo.initCartDB = function(){
    // open the database
    var shortName = 'Cart';
    var version = '1.0';
    var displayName = 'Cart';
    var maxSize = 65536;
```

Then the database is opened, and the variable `demo.db` is assigned a reference to the database.

```
// open the database
this.db = openDatabase(shortName, version, displayName, maxSize);
```

Finally, the database table that will hold the cart items is created, if it does not exist, using a syncronous database transaction.

```
    // create the cartItems table
    this.db.transaction(
        function(transaction) {
            transaction.executeSql(
                // SQL statement to create the cart table
                'CREATE TABLE IF NOT EXISTS cartItems ' +
                ' (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,' +
                ' item_id INTEGER, name TEXT, price REAL,
    ' quantity INTEGER);'
            );
        }
    );
}; //end of demo.initCartDB
```

The schema for the cart is straightforward. There is a column to store the id for each row that is autoincremented, three columns to store information about the product: item_id, name, and price, and a column to store the quantity of the product in the cart.

We will be intializing the database when the *index.html* page is opened. To do this, add a call to the `demo.initCartDB()` function to the jQuery document ready function, which is at the end of the file *site.js*.

```
$(function(){
    demo.loadCatalog();

    // initialize the database
    demo.initCartDB();

});
```

The next time the page is loaded, the HTML5 database `Cart` will be initialized, and the `cartItems` table will be created. If you are using Chrome or Safari, you can view the new table in the Storage section of the Developer Tools dialog.

Step 3: Store the Product Information in the cartItems Database

Now that the database is initialized, we need to define a function that will add the product to the cartitems table when the add to cart button is pressed. Replace the `demo.addToCart()` function created in "Step 1: Bind the Add to Cart Button to the addToCart() Function" on page 20 with the following code:

```
demo.addToCart = function(productId)
{
    // get the product from Taffy DB
    var product = demo.products.get({id:productId}).pop();

    this.db.transaction(
        function(transaction)
        {
            // check if a similar item is already in the cart with
            // same entity id and configuration
            transaction.executeSql('select id from cartItems where
item_id = ?', [product.id],

                function(transaction, result)
                {
                    if(result.rows.length > 0) // there is an
                                               // existing record
                    {
                        // increment the quantity of the existing
                        // record
                        var row = result.rows.item(0);
```

```
                        transaction.executeSql(
                            // SQL statement to increment the
                            // quantity of the item
                            'update cartItems set quantity = quantity +
1 where id = ?', [row.id],

                            function(transaction1, result1)
                            {
                                console.log('Quantity was incremented');
                                demo.showMessage('#product',
'This item is already in your cart. Its quantity has been increased by 1.');
                            },
                            demo.dbErrorHandler
                        );
                    }

                    else // there is not an existing record
                    {
                        // insert a new record
                        transaction.executeSql(
                            // SQL statement to insert the item
                            'insert into cartItems (item_id, name,
price, quantity) values (?, ?, ?, 1)', [product.id, product.name,
parseFloat(product.price, 10)],

                            function(transaction1, result1)
                            {
                                console.log('Item was inserted');
                                demo.showMessage('#product',
'This item was added to your cart.');
                            },
                            demo.dbErrorHandler
                        );

                    }
                },
                demo.dbErrorHandler
            );

        }
    );

};
```

Let's start with the three SQL statements that are defined. The first statement runs a query in the database to find an existing cart item using the product id.

```
transaction.executeSql('select id from cartItems where item_id = ?',
[product.id],
```

The result of the query is then passed to an anonymous function, which will check if the product has already been added to the cart by using the `length` attribute of the result rows.

```
if(result.rows.length > 0) // there is an existing record
```

If the value of result.rows.length > 0, then the product has already been added to the cart, and an update query is executed to increment the quantity of the product in the cart.

```
'update cartItems set quantity = quantity + 1 where id = ?', [row.id],
```

If the product has not been added to the cart, an insert query is executed.

```
'insert into cartItems (item_id, name, price, quantity) VALUES (?, ?,
 ?, 1)', [    product.id, product.name, parseFloat(product.price, 10)],
```

With these three statements, both inserts and updates can be handled when the `demo.addToCart()` function is called.

For each of the SQL transactions, the errors are handled by the `demo.dbErrorHandler()` function. This will display an alert when an error is encountered. Insert the following code after the `demo.addToCart()` function to add the `demo.dbErrorHandler()` function.

```
demo.dbErrorHandler = function(transaction, error) {
    // display an alert message if there is an error
    alert('Error: '+error.message+' [Code: '+error.code+']');
    return true;
};
```

After the insert or update statements are executed, a confirmation message is displayed to the user with the `demo.showMessage()` function. Insert the following code after the `demo.dbErrorHandler()` function to add the `demo.showMessage()` function.

```
demo.showMessage = function(viewId, message)
{
    // display the meesage text in the view for 3 seconds, then fade
out over 2 seconds
    $(viewId + ' .message').text(message).fadeIn().delay(3000).fadeOu
t(2000);
};
```

Update the message class in the *site.css* file to format the display of the message.

```
.message {
    display: none;
    width: 284px;
    padding: 4px;
    background-color: white;
    border: 1px solid orange;
}
```

Finally, update the `demo.setCurrentView` function to hide visible messages when a view is opened

```
demo.setCurrentView = function(viewId)
{
```

```
        $('#content > *').removeClass('current');
        $(viewId).addClass('current');
        $('.message').hide();
    }
```

Now, when the add to cart button is clicked, the product will be either added to the cart or the product's quantity will be incremented in the cart. If you are using Chrome or Safari, you can view the rows of the cartItems table in the Storage section of the Developer Tools dialog.

Step 4: Add the HTML for the Cart View and Bind the View Cart Button to the demo.viewCart() Function

Now that the products can be added to the Cart database, we need to allow the user to view the contents of the cart. Open the file *index.html* and insert the view card button into the product view.

```
<div id="product">
    <div class="message"></div>
    <nav>
        <button class="back">Back</button>
        <h1>Product Details</h1>
    </nav>
    <div class="product-details">
        <dl>
            <dt>Name</dt><dd class="name"></dd>
            <dt>SKU</dt><dd class="sku"></dd>
            <dt>Price</dt><dd class="price"></dd>
            <dt>Quantity</dt><dd class="quantity"><input type="text"
/></dd>
        </dl>
        <button class="add-to-cart">+ Add to Cart</button>
        <button class="view-cart">View Cart</button>
    </div>
</div>
```

Next, bind the view cart button to the `demo.viewCart()` function that will open the cart view. Open *site.js*, and add the following code to the end of the `demo.viewProduct()` function.

```
// bind the view cart button
$('#product .view-cart').unbind('click').click(function(e){
    e.preventDefault();

    // use HTML5 local storage to store the productId
    localStorage.setItem("lastProductId", productId);

    // display the cart view
    demo.viewCart();
});
```

Notice that the productId is stored using HTML5 local storage. We will use this id to allow the back button in the cart view to return the user to this product.

Next, add the HTML for the cart view. Insert the following code after the product view updated at the beginning of this step:

```
<div id="cart">
    <div class="message"></div>
    <nav>
        <button class="back">Back</button>
        <h1>Your Cart</h1>
    </nav>
    <div class="cart-item-list"></div>
    <form class="checkoutForm" method="POST" action="checkout.php">
        <input class="cartData" name="cartData" type="hidden"
value="" />
        <button  class="checkout-button" >Checkout</button>
    </form>

    <button class="clear-cart">Clear Cart</button>
</div>
```

Notice that there is a form with a hidden field and a button. We'll use this form later to send a stringified version of the cart to the server for order processing.

Step 5: Display the Cart

Now that the user can click the cart button to open the cart view, we need to get the cart items from cart database, and display the cart to the user. Open the file *site.js*, and insert the following code after the `demo.showMessage()` function.

```
demo.viewCart = function()
{
    // get the items from the db; demo.drawCart is provided as a call
back function
    demo.getCartItems(demo.drawCart);
};

demo.getCartItems = function(callback)
{
    this.db.transaction(

        function(transaction) {
            transaction.executeSql(
                // SQL statement to get all the items in the cart
                'SELECT * FROM cartitems order by id asc;', [],
                function(transaction, result) {
                    var cartItems = [];
```

```
                    for ( var i = 0; i < result.rows.length; i++) {
                        var row = result.rows.item(i);
                        cartItems.push(row);
                    }

                    callback(cartItems);
                },
                demo.errorHandler
            );
        }
    );
};
```

The `demo.viewCart()` function was bound to the view cart button in a previous step. This function will call the demo.getCartItems() function providing the demo.drawCart() function as the call back function parameter.

```
demo.getCartItems(demo.drawCart);
```

The demo demo.getCartItems() function will execute an SQL query to get all the rows from the cartItems table.

```
'SELECT * FROM cartitems order by id asc;', [],
```

The result rows are passed to an anonymous function, which will place the resut rows in an array named cartItems. cartItems is passed as a parameter to the call back function.

```
function(transaction, result) {
    var cartItems = [];
    for ( var i = 0; i < result.rows.length; i++) {
        var row = result.rows.item(i);
        cartItems.push(row);
    }
    callback(cartItems);
};
```

Next, insert the following code after the `demo.getCartItems()` function.

```
demo.drawCart = function(cartItems)
{
    demo.setCurrentView('#cart');

    // create the html for the cart
    var html = '<table>';

    // the column titles
    html += '<tr>';
    html += '<th>Name</th>';
    html += '<th>QTY</th>';
    html += '<th>Total</th>';
    html += '</tr>';
```

```
    // the rows for the items in the cart
    var total = 0;
    for(i in cartItems)
    {
        html += '<tr>';
        html += '<td>' + cartItems[i]['name'] + '</td>';
        html += '<td>' + cartItems[i]['quantity'] + '</td>';
        html += '<td>' + demo.formatStringAsCurrency(cartItems[i]
['price'] * cartItems[i]['quantity']) + '</td>';
        html += '</tr>';

        total += cartItems[i]['price'] * cartItems[i]['quantity'];
    }

    html += '<tr>';
    html += '<td> </td>';
    html += '<td> </td>';
    html += '<td class="total" >' + demo.formatStringAsCurrency(total)
+ '</td>';
    html += '</tr>';
    html += '</table>';

    // add the html to the view
    $('#cart .cart-item-list').empty().append(html);

    // add a stringified version of the cart to the checkout form
    demo.updateCartString();

    // bind the navigation links
    $('#cart .back').unbind('click').click(function(e){
        // display the product view
        demo.viewProduct(localStorage.getItem("lastProductId"));
    });

    $('#cart .clear-cart').unbind('click').click(function(e){
        // remove all the items from the cart
        demo.clearCartItems(demo.viewCart);
    });

};

demo.updateCartString = function()
{
   //Stubbed out for now. See Step #7
};
```

This function starts by opening the cart view created in "Step 4: Add the HTML for the Cart View and Bind the View Cart Button to the demo.viewCart() Function" on page 25.

```
demo.setCurrentView('#cart');
```

The function then defines an HTML string for the cart table that will be displayed to the user. The table has three columns: Name, Quantity, and Total.

These values are set using the values in the cartItems array. The local variable `total` is used to sum the price times the quantity for the cart items.

```
var total = 0;
for(i in cartItems)
{
    html += '<tr>';
    html += '<td>' + cartItems[i]['name'] + '</td>';
    html += '<td>' + cartItems[i]['quantity'] + '</td>';
    html += '<td>' + demo.formatStringAsCurrency(cartItems[i]['price']
* cartItems[i]['quantity']) + '</td>';
    html += '</tr>';

    total += cartItems[i]['price'] * cartItems[i]['quantity'];
}
```

The value of `total` is then added to the last row of the table.

```
html += '<td class="total" >' + demo.formatStringAsCurrency(total) +
'</td>';
```

Next, the HTML string is appended to the cart view, and the `demo.updateCartString()` function is called.

```
// add the html to the view
$('#cart .cart-item-list').empty().append(html);

// add a stringified version of the cart to the checkout form
demo.updateCartString();
```

A placeholder for the `demo.updateCartString()` function has been provided as the `demo.drawCart()` function. The contents of the `demo.updateCartString()` function will be added in "Step 7: Stringify the Cart to Send to the Server" on page 30 in order to insert a stringified version of the cart into the checkout form.

Next, the back button is bound to the `demo.viewProduct()` function. Notice that the product id passed into the function is the value from the HTML5 local storage that was set in "Step 4: Add the HTML for the Cart View and Bind the View Cart Button to the demo.viewCart() Function" on page 25.

```
demo.viewProduct(localStorage.getItem("lastProductId"));
```

Finally, the clear cart button is bound to the `demo.clearCartItems()` function, which will be added in "Step 8: Clear the Cart Contents" on page 31. The `demo.viewCart()` function is provided as the call back function parameter in order to redraw the cart view after the cart items are removed.

```
demo.clearCartItems(demo.viewCart);
```

You will now see the contents of the cart when the cart view is opened.

Step 6: Format the Cart View

Add the following code to the end of the file *site.css* to format the display of the cart table and related buttons:

```
#product .view-cart {
    margin-top: 20px;
    width: 300px;
}

#cart table {
    border-left: 1px solid silver;
    border-top: 1px solid silver;
    width: 300px;
}

#cart td, #cart th {
    border-right: 1px solid silver;
    border-bottom: 1px solid silver;
    padding: 4px;
    font-size:14px;
}

#cart td:nth-child(2) {
    text-align: center;
}

#cart td:nth-child(3) {
    text-align: right;
}

#cart .checkout-button, #cart .clear-cart {
    margin-top: 20px;
    width: 300px;
}
```

Step 7: Stringify the Cart to Send to the Server

Now that the cart contents are displayed to the user, we need to provide a way to send the cart to the server for order processing. To do this, we will stringify the SQL result produces by the `demo.getCartItems()` function. Replace the `demo.updateCartString()` function with the following code:

```
demo.updateCartString = function()
{
    // get the cart items from the db; demo.stringifyCart is provided
    // as a callback function
    demo.getCartItems(demo.stringifyCart);
};

demo.stringifyCart = function(cartItems)
```

```
{
    // stringify the cart data, and add the string to a hidden value
    // in the checkout form
    $('#cart form.checkoutForm .cartData').val(JSON.stringify(cartIte
ms));
};
```

The `demo.updateCartString` calls the `demo.getCartItems()` function providing the `demo.stringifyCart()` function for the call back funtion parameter. This will get all the items in the cartItems table, and pass them to the `demo.string ifyCart()` function.

The `demo.stringifyCart()` function will use the `JSON.stringify` function to convert the cartItems array into a string, and then the string will be inserted as the value for the hidden form field with the class cartData in the checkout form. When the user clicks the checkout button, the stringified cart will be submitted to the server for order processing.

The tutorial directory contains the file *checkout.php*, which will display the contents of the posted checkout form.

Step 8: Clear the Cart Contents

For the final step, we will add a function that will remove all the items from the cartItem table when the clear cart button is clicked. Add the following code to the file *site.js* after the `demo.stringifyCart()` function.

```
demo.clearCartItems = function(callBack)
{
    this.db.transaction(
        function(transaction) {
            transaction.executeSql(
                // SQL statement to delete all the items in the cart
                'delete from cartItems', [],
                function(){
                    callBack();
                },
                demo.errorHandler
            );
        }
    );
};
```

This will execute an SQL statement to delete all the items from the table.

That's It

You should now have a simple working mobile commerce site. Enjoy.

CHAPTER 3

Analyzing Open Data for Fun (and Profit?), Part 1

Abe Music

With the ability to enhance the the way that we do business, transform the way we govern ourselves, and increase the visibility that we have on important aspects of life that often go overlooked, such as the environment and poverty levels, it's no wonder that open data has stirred up so much buzz. While software might be the first thing that comes to mind when you hear the term "open source," hopefully it's not the last. Sure, the software is a phenomenon in and of itself, but in the end, software is always the enabler for something else in the real world. It's the open source movement's underlying philosophy that promotes transparency, community, and pragmatic methodology that's the real phenomenon, having produced diverse projects that range from open hardware to open cola to open government.

The remainder of this article and its continuation Chapter 4 explores the process of building a simple Java- and JavaScript-based infrastructure based on the Google Chart Tools to ingest and visualize an interesting data set recently opened up by the World Bank. This is an incredibly rich set of data and includes everything from the number of tractors in Afghanistan to China's urban population to various economic and financial indicators. In addition to being an interesting set of data for demo purposes, there are definitely possibilities for ways that you could mine this data set and sell the analysis as a report or perhaps in tiny bite-sized chunks using a micropayments sales strategy. We'll investigate a few of these possibilities in a future article. In the same way that services-based business models emerged around open source software, we're very likely to see an increasing number of business models evolve around open data. I hope you're as excited about these possibilities as I am!

Getting the World Bank Data

The World Bank recently launched an online data catalog that houses an enormous amount of interesting information. One of the more popular items in the catalog is the World Development Indicators database. The World Bank's description of the database states, "The World Development Indicators (WDI) provides a comprehensive selection of economic, social and environmental indicators, drawing on data from the World Bank and more than 30 partner agencies. The database covers more than 900 indicators for 210 economies with data back to 1960." Wow, that's a lot of data—so much that it might be hard to figure out what to even do with it. Fortunately, the data is available in a number of highly accessible formats, including a comma separated values (CSV) file and open API, so the barrier to entry is very low.

Because we'll be serving up the data to Ajax-based visualization widgets and the World Bank APIs don't yet support any cross-domain mechanisms such as JSONP (JSON with Padding), it'll be convenient to have full access to the data on the same domain as the widgets. So, head over to the data catalog and download the CSV-formatted files for the Worldwide Development Indicators data set. Take a moment to peruse the files when you download them, but be careful in opening them since some are quite large. The file we're primarily interested in is the largest one, *WDI_GDF_Data.csv*, which is over 100 megabytes in size.

You can peek at the layout of the file on a *nix system by using the head command:

```
$ head -2 WDI_GDF_Data.csv

Series Code,Series Name,Country Code,Country Name,1960,...,2009
AG.AGR.TRAC.NO,"Agricultural machinery, tractors",AFG,Afghanistan,,
1.20000000000000E+02,...,,
```

Thus, there are four columns that denote a series and country combination, with the remaining columns providing data points for each year in the series. For example, the sample output shows that Afghanistan had 120 tractors on file for the year 1960. With the data in hand, the next step is to load it into MySQL where we can easily slice it, dice it, and serve it up to rich visualizations in the browser.

Loading the Data into MySQL

If you don't already have MySQL installed, the MySQL website offers extensive documentation on configuring MySQL, and it's assumed that you're familiar enough with MySQL to login to the console, create a table in a database, and

perform basic queries. The process of importing CSV data into MySQL is a two-step process: creating a table for it and then actually loading the CSV file. Executing the following command performs both steps, which creates a mydb (My Database) database and a table called wdi_gdf (for World Development Indicators and Global Development Finance.)

$mysql -u user -p < load.mysql

```
create database mydb;

use mydb;

create table wdi_gdf (
    series_code varchar(256),
    series_name varchar(256),
    country_code varchar(256),
    country_name varchar(256),
    y1960 float,
    /* Truncated program listing. See the Resources section
       of this article for a full program listing (load.mysql)*/
    y2009 float)
engine = myISAM;

/* NOTE: Change the path to the csv file below to wherever you
   put it on your machine */
load data infile '/tmp/WDI_GDF_Data.csv' into table wdi_gdf
fields terminated by ',' enclosed by '"'
lines terminated by '\r\n'
ignore 1 lines;

create index idx_series_name on wdi_gdf (series_name);
create index idx_country_name on wdi_gdf (country_name);
```

The given schema of the data lends itself to SQL queries of the following form: SELECT <year,...,year> FROM <table> WHERE country_name = <country> and series_name = <series>, so we've added indices on these fields for speedy lookup. Conceptually, the result set returned from such a query is a single tuple of numbers where each number corresponds to each year of interest. As we'll see later, the format of these result sets is potentially awkward and may need to be transposed depending on the expectation of the visualization widget, but the additional logic required to massage the data is relatively straightforward. As a word of advice, it's generally best to err toward simplicity when first exploring a data set in a relational database since you don't know much about the data or how you'll ultimately end up using it. It's easy enough to carve out a schema once you've determined that the investment is worth your time. (An interesting alternative to loading the data into a relational database for the WDI data is to use a schema-less system such as a key-value store, but that's outside the scope of this article.)

Downloading Google Visualization Tools

The Google Chart Tools provide server- and client-side tools. The server-side tools include example infrastructure for reading CSV files, MySQL databases, and custom data sources implementing the Wire Protocol. The most interesting part of the Wire protocol is that it defines a tabular format for data that's served to a client such as a visualization widget. The server-side tools also provide the processor for a simplified SQL-like query language, which takes care of sanitizing query requests and fetching data from an underlying database. The client-side tools are JavaScript libraries for retrieving data from a variety of server-side data sources such as those just described, Google Spreadsheets, and any other source emitting data in the tabular format defined the the Wire Protocol.

Download the latest stable server-side visualization tools online at Google Code. Currently, version 1.0.2 of the visualization tools have pre-built *jar* files ready for use. If for some reason you wish to build from source you can checkout the source code from the project page and use Maven or Ant. While this article does not require extensive knowledge of building Java-based web apps, it does assume basic familiarity with Java and servlet-based technologies. The example of interest for this article is the SqlDataSourceServlet class, which implements the following method:

```
public DataTable generateDataTable(Query query, HttpServletRequest
request) throws DataSourceException {
    SqlDatabaseDescription dbDescription = new SqlDatabaseDescription(
        request.getParameter("url"),
        request.getParameter("user"),
        request.getParameter("password"),
        request.getParameter("table")
    );
    return SqlDataSourceHelper.executeQuery(query, dbDescription);
}
```

In short, query requests that are passed in include values for url, user, password, and table to establish a database connection. (Depending on your specific situation, you might need to implement a more secure mechanism than passing in these values via client-side request for any situation other than basic development.) Once the connection is established, the query is executed and the results are encoded as a DataTable returned back to the client. Note that the URL parameter for the query (defined as tq) has already been picked off of the parameter list, decoded, and instantiated into a Query object by the time generateDataTable is called.

When you're ready, all that's left before queries can be run and visualized is a basic installation of a web app in Tomcat.

Configuring a Visualization Web App with Tomcat

Like MySQL, Tomcat can be installed with your favorite package manager or you can download the project and install it manually. Extensive documentation is available online should you wish to alter the default configuration. The directory that Tomcat is installed into is generally referred to as *$CATALINA_HOME*, and once Tomcat is installed, you should be able to start it up by running $CATALINA_HOME/bin/startup.sh. Likewise, the shutdown.sh script stops the server. Log files are available in *$CATALINA_HOME/logs* should you need to debug your installation. With Tomcat running, you should be able to access *http://localhost:8080/* and get back a nice splash page confirming that your server is running. The remainder of this section provides very abridged instructions for building a sample web app and deploying the visualization code you built in the previous section into it. More thorough instructions are available online and won't be regurgitated here.

The *$CATALINA_HOME/webapps* directory contains either *WAR* files (web application archives) or exploded directory structures resembling those of *war* files. The directory structure of a standard web app like the one we'll be building follows:

```
$CATALINA_HOME/
    webapps/
        myWebApp/
            dynamic-content.jsp
            WEB-INF/
                web.xml
                lib/
                    lib1.jar
                    lib2.jar
```

A few brief notes about Java-based web applications may be helpful:

- The *web.xml* file provides URL-to-servlet mappings and other configuration information for a web application
- Dynamic content such as JSPs (Java Server Pages) resides in the web application root directory
- Libraries that are referenced by any Java code that executes needs to be in lib if it's not already in the $CLASSPATH

Create a basic directory structure for your web app by creating a *myWebApp/WEB-INF/lib* directory hierarchy in the webapps directory. Then, copy the following files into the lib directory from your visualization tools:

- visualization-datasource–1.0.2.jar
- visualization-datasource-examples.jar

- lib/*.jar (commons-lang–2.4.jar, commons-logging–1.1.1.jar, etc.)
- A MySQL connector that you'll need to acquire from *http://www.mysql.com/products/connector/j/*

Finally, you'll want to copy all of the files from *examples/src/html/* directly into your *myWebApp* directory except for the *web.xml* file which should go in your *WEB-INF* directory. At this point, you should verify that you can access the URL *http://localhost:8080/myWebApp/all_examples.html* and get back a page with various interactive visualizations on it.

Implementing and Querying a SQL Data Source

With the examples running in your web app, and MySQL loaded with the WDI data, the only thing standing between you and Wire Protocol powered queries served over HTTP are a couple of servlet mappings. Add the following mappings to your *web.xml* file and then restart Tomcat to make sure that the mappings are recognized. (Note that SqlDataSourceServlet is included in the files we just copied into the WEB-INF/lib directory.)

```
 <servlet>
   <servlet-name>SqlDataSourceServlet</servlet-name>
   <description>SqlDataSourceServlet</description>
   <servlet-class>SqlDataSourceServlet</servlet-class>
 </servlet>

<servlet-mapping>
   <servlet-name>SqlDataSourceServlet</servlet-name>
   <url-pattern>/sql</url-pattern>
</servlet-mapping>
```

After this mapping is established, all queries passed in via *http://localhost:8080/myWebApp/sql* are routed to the SqlDataSourceServlet, which fetches data from MySQL. Try out the following example query by typing it into your browser's URL bar to verify that you're properly up and running. Note that it's required to pass in properly escaped URLs, but in most situations, your browser should automatically escape the URL for you.

You may need to substitute correct parameters for your database connection in the URL via the name, password, and table variables.

http://localhost:8080/myWebApp/sql?tq=select y1960 where series_name="Urban population" and country_name="China"&url=jdbc:mysql://localhost:3306/mydb&user=user&password=password&table=wdi_gdf

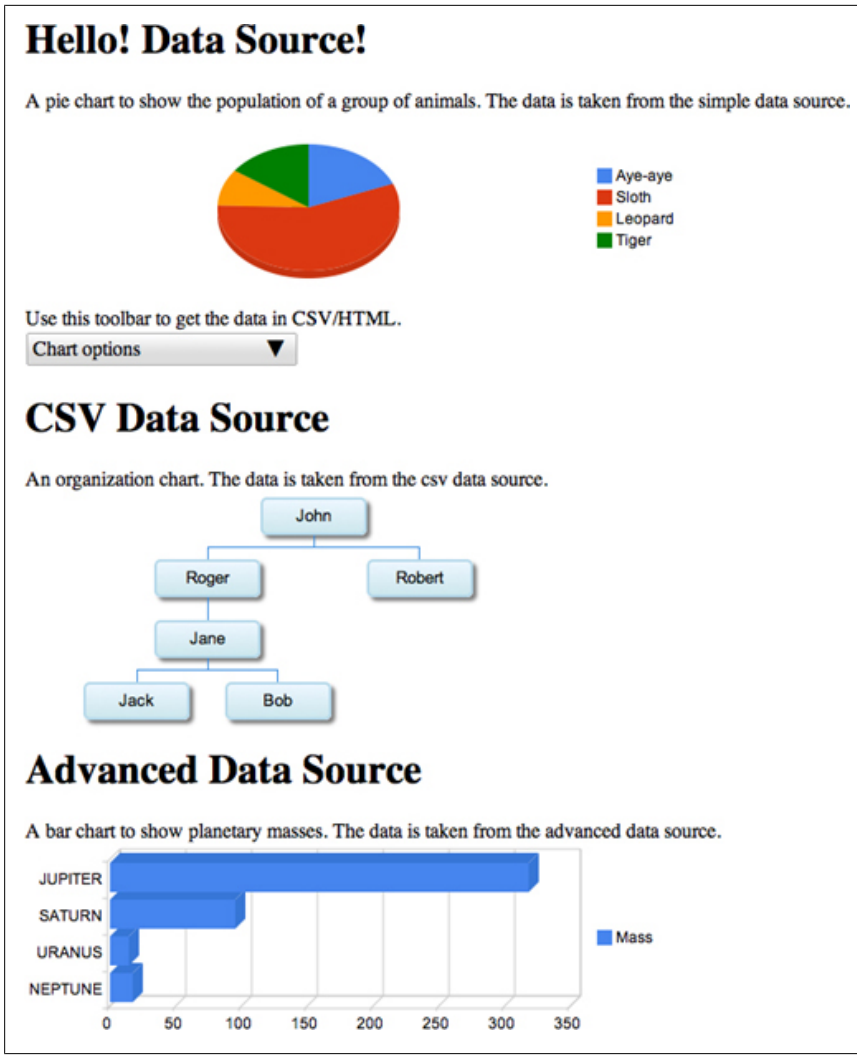


Figure 3-1. Sample visualizations that come with the Google visualization tools

Picking apart the query reveals the parameters expected by the SqlDataSourceServlet as well as the tq parameter that defines a SQL-like query. Complete documentation for the query language is online, but the gist is that it supports the most commonly used SQL syntax, including basic aggregate functions, limits, offsets, and ordering. Obviously, queries of the form DROP TABLE <table> are sanitized just as would happen with unsupported syntax such as SELECT DISTINCT <column>. The response to the sample query should be something that resembles the following output:

```
google.visualization.Query.setResponse({version:'0.6',status:'ok',
sig:'1133135268',table:{cols:[{id:'y1960',label:'y1960',type:'number',
pattern:''}],rows:[{c:[{v:1.06731E8}]}]}});
```

Pretty-printing the JSON portion of the response through a utility such as *http://jsonlint.com* makes it a bit more understandable.

```
{
    version: '0.6',
    status: 'ok',
    sig: '1133135268',
    table: {
        cols: [
            {
                id: 'y1960',
                label: 'y1960',
                type: 'number',
                pattern: ''
            }
        ],
        rows: [
            {
                c: [
                    { v: 1.06731E8 }
                ]
            }
        ]
    }
}
```

The important part of the response is the table field that defines columns and row values for each of those columns. As you'll see in the article, we'll have to massage the data in this structure so that the Google Chart Tools visualizations can consume it, but a client-side wrapper provides an intuitive API for manipulating the structure. All in all, it's not very difficult work and keeps your JavaScript chops sharp.

Next Steps...

That's all for this first installment, but in the follow on to this article, we'll take a look at how to implement table and line chart visualizations as well as respond to user events using the foundation you've established. A third and final installment will look at ways to monetize this app using PayPal X payment flows.

All sample code for this article series is available here.

CHAPTER 4

Analyzing Open Data for Fun (and Profit?), Part 2

Abe Music

This is the the second article in a series of articles that takes a look at using open source technology to analyze the World Development Indicators data set from the World Bank's data catalog.

Implementing Table and Line Chart Visualizations

The bad news is that it's taken a little bit of work to put together the scaffolding needed to get our data into a format that's easily consumed on the front end. The good news is that most of this effort is a one-time expense to get infrastructure in place and learn the ropes; the scaffolding we have in place is pretty robust and generalizes pretty well to any kind of tabular data that you might run across. The next step is to actually start visualizing it, and this is where things get a lot more interesting. At this point, it might be helpful to visit Google's Visualization Playground to get a basic idea of the various options available and peruse some sample code.

The steps involved in visualizing a query are generally the same: construct the query, run it, and pass the results into a visualization for rendering. The simplest visualization to start out with is a table, since it requires simply passing the result of the query into a Table constructor function. To see it in action you can copy the table1.jsp code into the *myWebApp* directory, modify the database connection variables, and point your browser to *http://localhost:8080/myWebApp/table1.jsp*. The following program listing demonstrates:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Table</title>

        <!-- Google JS lib -->
        <script type="text/javascript" src="http://www.google.com/jsapi">
</script>

        <script type="text/javascript">

            /* Load the Google visualization packages being used */
            google.load('visualization', '1', {'packages':['table']});

            google.setOnLoadCallback(function () {
                // Known constants for the data set
                var FIRST_YEAR = 1960;
                var LAST_YEAR = 2008;

                /* Modify the following variables to fit your
                environment */
                var DB_URL = "jdbc:mysql://localhost:3306/mydb";
                var DB_USER = "user";
                var DB_PASSWORD = "password";
                var DB_TABLE = "wdi_gdf";

                var years = ["y1960","y1970","y1980","y1990","y2000"];

                // Build a query to get data by year for a given
                // series and country
                var query = new google.visualization.Query(
                        'sql?tq=' +
                        'select ' + years.join(",") +
                        ' where series_name = "Urban population" and
country_name = "China"' +
                        "&url="+ escape(DB_URL) +
                        "&user="+ DB_USER +
                        "&password="+ DB_PASSWORD +
                        "&table="+ DB_TABLE
                );

                // Send the query to the server and run callback
                query.send(function(response) {
                    if (response.isError()) {
                      alert('Error in query: ' + response.getMessage()
+ ' ' + response.getDetailedMessage());
                      return;
                    }

                    var table = new google.visualization.Table(
document.getElementById('viz'));
                    var data = response.getDataTable();
```

```
                    // Formatting isn't required but dramatically
                    // increases readability of cell data
                    var formatter = new google.visualization.TableNumber
Format();
                    for (var c=0; c< years.length; c++) {
                        formatter.format(data, c); // Format each column
                    }

                    table.draw(data);
                });
            });
        </script>
    </head>
    <body>
        <div id="viz" style="width:600px"></div>
    </body>
</html>
```

y1960	y1970	y1980	y1990	y2000
106,731,000.00	142,387,000.00	192,322,000.00	311,041,000.00	452,027,000.00

Figure 4-1. Rendering your query as a sortable data table is incredibly easy

Hopefully, the code is pretty easy to follow from top to bottom. The visualization package is loaded for a table, and then a callback function executes a query and constructs a Table. Most of the code is a template that we can expand upon for more complex visualizations. From here on out, you could spend almost all of your time time coding and analyzing since you have an accessible data pipe and the ability to query it. The one exception is that you might need to restart Tomcat with more heap space if you begin to run very large queries or add additional indices if you'd like to query by series code or country code for some reason.

To demonstrate the Table visualization's API, let's expand upon the table example by writing a simple transpose function that will swap the rows and columns. As it turns out, this transformation is necessary for rendering a line chart in just a moment. The following code snippet illustrates the transpose function that is used in example table2.jsp.

```
var transposeData = function (data) {
    var newData = new google.visualization.DataTable();

    newData.addColumn("string", "Year");
    newData.addColumn("number", "Urban Population for China");

    for (var c = 0; c < data.getNumberOfColumns(); c++) {
        newData.addRow([{v : data.getColumnId(c)}, {v :
data.getValue(0, c) }]);
    }
```

```
    return newData;
}
```

Year	
y1960	106,731,000.00
y1970	142,387,000.00
y1980	192,322,000.00
y1990	311,041,000.00
y2000	452,027,000.00

Figure 4-2. Transposed query data rendered in rows instead of columns

From here on out, very minimal changes to the working template code are needed to produce common visualizations such as line charts, bar charts, maps, etc. For example, visualizing a line chart instead of a table only requires adding a reference to the linechart package in the `google.load` function and then making the following substitution. See linechart.jsp for the full program listing.

```
// Remember to make this update:
google.load('visualization', '1', {'packages':['linechart']});

// And here's how you create it
var data = transposeData(response.getDataTable());
var linechart = new google.visualization.LineChart(
document.getElementById('viz'));
linechart.draw(data);
```

That was pretty nifty, eh? But what about charting multiple populations on the same chart? Simply update the query to collect the additional country/series information and then make an update to the transpose function to account for the change (a wonderful exercise to sling some custom code of your own.) The line chart itself does the rest of the work for you.

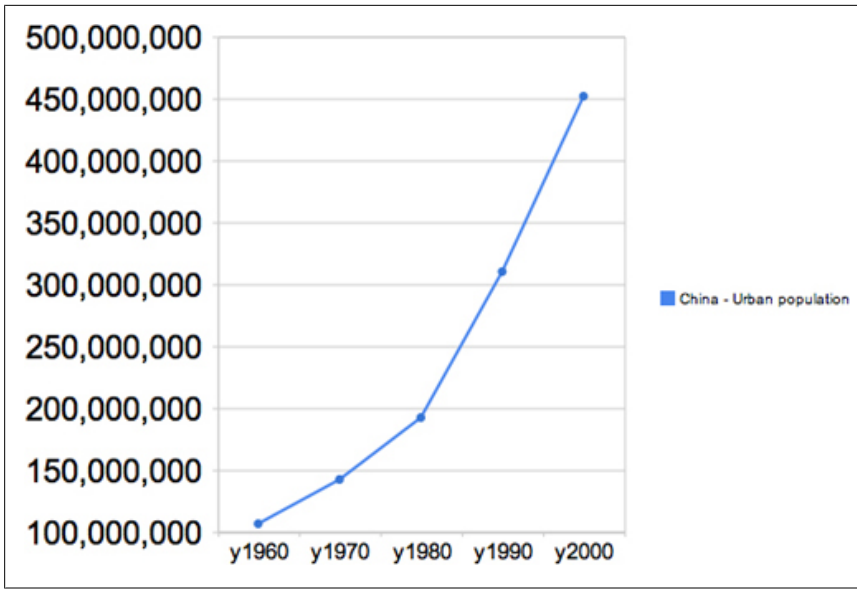


Figure 4-3. An example line chart rendered from the same data object as the table in Figure 3

Creating an Event-Driven Line Chart Application

Once you've hacked on the data long enough to determine that it's interesting and have enough example code in place that you can do something useful with it, the next step is automating a workflow so that you don't need to modify your code for every change you'd like to make. For the examples we've been working through, a nice workflow would be the ability to select a country and series combination from combo boxes and have the visualization automatically update to reflect the new data. Heck, that's the kind of thing that you might even want to share!

Combo boxes can be produced by writing standalone JSPs that can be inlined into the page. The JSPs will simply connect to MySQL, query the country or series information, and emit the expected markup by iterating over the result set. The following code listing shows the full JSP for creating the SELECT elements. The following listing demonstrates:

```
<%
// country_options.jsp
try {
    /* Modify the following variables to fit your environment */
    String DB_URL = "jdbc:mysql://localhost:3306/mydb";
    String DB_USER = "user";
    String DB_PASSWORD = "password";
    String DB_TABLE = "wdi_gdf"

    String query = "select distinct country_name from " + DB_TABLE +
" order by country_name";
    Class.forName("com.mysql.jdbc.Driver");

    java.sql.Connection conn = java.sql.DriverManager.getConnection(
DB_URL, DB_USER, DB_PASSWORD);
    java.sql.Statement stmt = conn.createStatement();
    java.sql.ResultSet rs = stmt.executeQuery(query);

    while (rs.next()) {
        out.println("<option>" + rs.getString("country_name") +
"</option>" );
    }

    rs.close();
    stmt.close();
    conn.close();
}
catch (java.sql.SQLException ex) {
    while (ex != null) {
        System.out.println("SQL Exception:  " + ex.getMessage ());
        ex = ex.getNextException ();
    }
}
catch (java.lang.Exception ex) {
    System.out.println("java.lang.Exception:  " + ex.getMessage ());
}
%>
```

The logic for the series-based element works exactly the same except that the basis of the query is series_name instead of country_name. Inlining these JSPs into our existing template compartmentalizes the code and eases the maintenance burden; it's as simple as this:

```
<select id="country_name">
    <!-- Use a JSP to build up the options -->
    <%@ include file="./country_options.jsp" %>
</select>
```

Once you create the SELECT elements, one thing will quickly become apparent: there is so much data in them that they're burdensome to use. JavaScript toolkits offer ComboBox widgets that can trivially transform any SELECT element into components with autocomplete and filtering capabilities, so let's use one of those. Dojo's ComboBox widget sounds like a good choice for this

kind of job, but you could just as easily integrate any other widget that fits the bill.

The basic steps involved in integrating the ComboBox widget are building up the SELECT elements with all of the appropriate OPTION tags from the JSP, loading Dojo, instantiating the SELECTs as ComboBoxes, and finally, wiring up some event handlers to reload the visualization when a selection in one of the ComboBoxes changes.

Loading Dojo or other Ajax toolkits is trivial since Google has integrated it into their JavaScript infrastructure. The following snippet demonstrates a couple of SCRIPT blocks with the bare minimums required to load Dojo (and a ComboBox) from Google's CDN (Content Delivery Network). Consult DojoCampus.org for extensive online documentation and examples about Dojo. What follows is a template for integrating Dojo into our working example code:

```
<script type="text/javascript" src="http://www.google.com/jsapi">
</script>

<script type="text/javascript">
    /* Use Google's CDN to load Dojo, parse widgets, and handle page
    load */
    djConfig={
        require : ["dijit.form.ComboBox"], // Pull in the resources/
                                           // widgets needed
        parseOnLoad: true,                 //Parse widgets once Dojo
                                           // is ready
        addOnLoad : function() {
            // Do something interesting in here with Dojo. Don't reference
            // dojo.* or dijit.* anywhere else until you are are sure that
            // the page has loaded or you'll have a race-condition on
            // your hands
        }
    };
    google.load("dojo", "1.5.0");
```

With Dojo at our fingertips, all that's left at this point is instantiating the ComboBoxes from the SELECT elements and creating the event handlers to reload the visualization whenever a value changes. The approach we'll take is to add a dojoType="dijit.form.ComboBox" attribute in the SELECT tag and use the dojo.connect function to listen for changes. That might sound like a lot to swallow, but it's actually pretty simple and makes a lot more sense when you see it. An abbreviated program listing for wdi_gdf.jsp follows:

```
<!DOCTYPE html>
<html>
    <head>
  <title>World Development Indicators</title>
    <!-- Stylesheets for Dojo widget/look-feel -->
```

```
    <style type="text/css">
        @import "http://ajax.googleapis.com/ajax/libs/dojo/1.5.0/dojo
/resources/dojo.css";
        @import "http://ajax.googleapis.com/ajax/libs/dojo/1.5.0/dijit
/themes/tundra/tundra.css";
    </style>

    <!-- Google JS lib -->
    <script type="text/javascript" src="http://www.google.com/jsapi">
</script>

    <script type="text/javascript">

        /* Load the Google visualization packages being used */
        google.load('visualization', '1', {'packages':['linechart']});

        /* Use Google's CDN to load Dojo, parse widgets, and handle
        page load */
        djConfig={
            parseOnLoad: true,
            require : ["dijit.form.ComboBox"],
            addOnLoad : function() {
                // Reload chart whenever combo boxes change
                dojo.connect(dijit.byId("country"), "onChange",
loadSeriesForCountry);
                dojo.connect(dijit.byId("series"), "onChange",
loadSeriesForCountry);

                //Set a default
                dijit.byId("series").attr('value', 'Urban population');
                dijit.byId("country").attr('value', 'China');
            }
        };
        google.load("dojo", "1.5.0");

        function loadSeriesForCountry() {
            /* See wdi_gdf.jsp for full program listing */
        }
    </script>
</head>
<body class="tundra" style="padding:20px">
    Select a country and series to plot the World Bank's
    <a href="http://data.worldbank.org/data-catalog/world-development
-indicators">
        World Development Indicators and Global Development Finance
    </a> data.
    <table>
        <tr>
            <td>Country:</td>
            <td>
                <!-- Use a JSP to build up the options and use Dojo
for auto-complete/filtering-->
                <select id="country" dojoType="dijit.form.ComboBox"
style="width:800px">
```

```
                    <%@ include file="./country_options.jsp" %>
                </select>
            </td>
        </tr>
        <tr>
            <td>Series:</td>
            <td>
                <!-- Use a JSP to build up the options and use Dojo
for auto-complete/filtering-->
                <select id="series" dojoType="dijit.form.ComboBox"
style="width:800px">
                    <%@ include file="./series_options.jsp" %>
                </select>
            </td>
        </tr>
    </table>
    <div id="viz"></div>
</body>
</html>
```

With convenient combo boxes in place, you now have a bona fide mini-application on your hands that can plot a line chart for any country/series combination of your choosing.

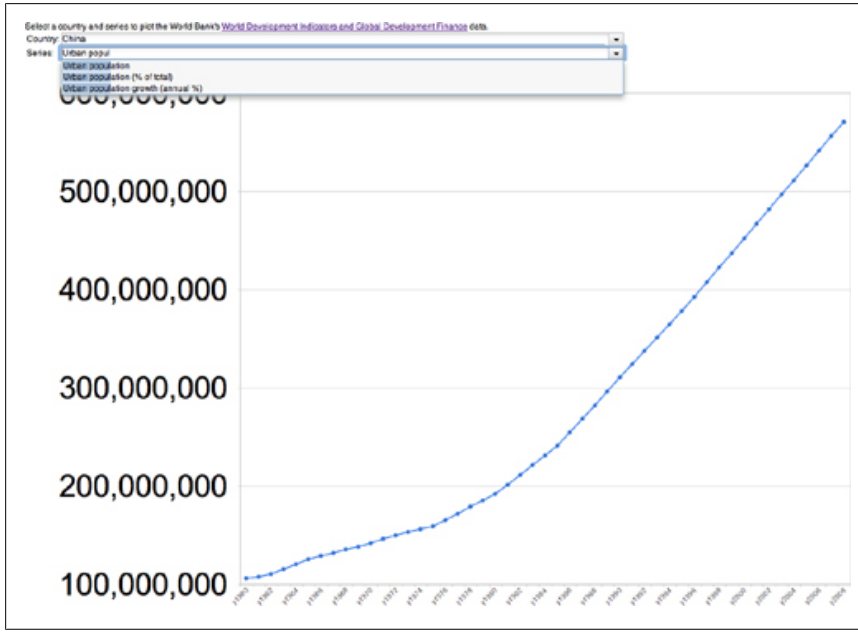


Figure 4-4. A mini-application consisting of a dynamic line chart driven by combo boxes that filter user input

Next Steps

We covered a serious amount of ground in this short article series, and you're now equipped with the tools and know-how to grab some interesting data, load it into a convenient infrastructure, and make sense of it using powerful visualizations that you can easily share with others via the Web. There are many interesting things you could do at this point. A few recommendations for next steps include:

- Plotting multiple data series on the same Line Chart
- Rendering on the GeoMap visualization
- Using animated visualizations such as the Motion Chart to show changes over time
- Implementing custom event handlers to provide additional user interactions

The Visualization Playground provides some sample code for all of their visualizations, and with the infrastructure you already have in place, your imagination should be the only limiting factor.

Until Next Time

In the third and final installment to this article series, we'll discuss some possible ways that you can monetize open data using PayPal X payment flows. In the meanwhile, have some fun hacking and visualizing the World Bank's WDI data set and try to come up with some nice visualizations.

All sample code for this article is available here.

CHAPTER 5

Analyzing Open Data for Fun (and Profit?), Part 3

Abe Music

This is the third and final installment in a series of articles that takes a look at using open source technology to analyze the World Development Indicators data set from the World Bank's data catalog. In this final article, we're going to change gears a bit and look at one possible way we might monetize our analysis efforts thus far by leveraging the PayPal X payment platform. Just in case you missed the first two articles in the series, click here for Part 1 and here for Part 2.

PayPal X: Your Global Payment Platform

PayPal X, the latest branding of the PayPal ecommerce platform, introduces several new payment APIs to make your life as a developer wishing to collect payments through your site or application easier and more robust. The traditional payment gateways, such as Express Checkout and Web Payments, still exist; however, those products were mainly used in your typical (read boring) web checkout routines. With the new Adaptive Payments API you have finer-grained control over how the payments are sent and received. You could split payments among multiple recipients, offer preapproved payments, and process payments on any number of platforms! Another great new feature offered by the Adaptive Payments API is Embedded Payments, which allows you to embed PayPal payments directly in your application without redirecting away from your site. The possibilities are endless, and in this article we're going to take a look at Embedded Payments and how we can use it to turn our already built analysis application into a revenue generating machine.

For a more thorough overview of Adaptive Payments and what is possible for developers you should really explore the PayPal X site or jump right in and grab a PDF of the Adaptive Payments documentation.

Dollars and Sense

Before we jump in, let's take a second and think about what we've done so far and how a payment system might fold into the mix. The first questions that might come to mind are (a) what are we going to offer our consumers and (b) at what price? So far we have a nifty application that dynamically generates a chart given a country and a world development indicator (WDI). Remember that this data is pulled from a MySQL database generated from the WDI portion of the World Bank's data catalog. This doesn't seem like something a user would pay for, but for the purposes of this article let's imagine a scenario where users would want to pay for the byproducts of our analysis. Some reasons could be:

- the dataset is proprietary or not free to the public
- the dataset is too large or complex and would require a significant investment in computing resources (e.g., mining millions of documents could take dozens of computers working in parallel to produce results in a timely manner)
- or maybe it's simply convenient to use and offered at a reasonable price

Well, it looks like the last option fits the bill quite nicely, so let's move forward with that assumption. Now that we know what we're offering our customers and why they would want to purchase our wares we need to figure out a pricing model. We could offer a monthly subscription for unlimited use as long as the user's account is in good standing, or we could require a user to purchase each and every visualization. That doesn't sound too convenient (or fun) at all. What about virtual currency? With major players, like Google and Facebook, offering up their own virtual currencies, we might as well cash in on the concept too (no pun intended!)

In case you're not familiar with the concept of virtual currency, it's a lot like the tokens you buy at video arcades that have no real value except to play video games in that one arcade. The tokens in this case are digitally tied to a user's account for purchasing virtual goods in an online application like Zynga's very popular Farmville. For our analysis application, we're going to provide a virtual currency called V-Rupees and each V-Rupee will allow a customer to purchase one visualized graph.

Sounds silly doesn't it? Well, there are benefits of virtual currencies that aren't immediately obvious. One benefit is that you get the money up front, and the user must spend the virtual currency in your application. Another benefit is that the number of real dollar transactions may be significantly lower resulting in less transaction fees by payment gateways like PayPal. For example, suppose you were selling a digital good for $1.00, with PayPal's 2.9% + $0.30 transaction fee, you would be hit with $0.33 worth of fees for every unit sold. Using the same transaction fee, suppose a user bought 5 units upfront. You would only be charged $0.45. That's a savings of $1.20! Not all users would want to buy in bulk; however, by providing incentives to purchase more units up front, you could reduce your transaction fees by a considerable amount.

It should be noted that the transaction fee for traditional PayPal merchant accounts was used in the calculations above. PayPal now offers a micropayment merchant account that is tuned for merchants processing low-value (less than $12) transactions with a rate of 5% + $0.05. If you are selling goods with a low dollar amount you should definitely check it out!

	Transaction Fees per Account Type		
Amount	Standard	Micropayment	Savings
$1.00	$0.33	$0.10	$0.23
$2.00	$0.36	$0.15	$0.21
$3.00	$0.39	$0.20	$0.19
$4.00	$0.42	$0.25	$0.17
$5.00	$0.45	$0.30	$0.15
$6.00	$0.47	$0.35	$0.12
$7.00	$0.50	$0.40	$0.10
$8.00	$0.53	$0.45	$0.08
$9.00	$0.56	$0.50	$0.06
$10.00	$0.59	$0.55	$0.04
$11.00	$0.62	$0.60	$0.02
$12.00	$0.65	$0.65	$0.00

Figure 5-1. Transaction fees for standard vs micropayments

Back to our V-Rupees. One unit of our virtual currency is equivalent to $0.01, but to reduce the transaction fees we will impose a minimum purchase of 100 V-Rupees and a maximum of 500. Like the example above, this is a very low dollar amount, so we should use a PayPal micropayment account. The micropayment transaction fee for buying 100 V-Rupees is $0.10 (compared to $0.33 for the traditional PayPal fee.)

Adaptive Payments Configuration

To start, we need to configure our existing webapp (myWebApp) to use the Adaptive Payments API. It's beyond the scope of this article to explain all the gory details about the configuration, but you'll be pleasantly surprised to find a multitude of other articles available at the PayPal X DevZone. Here's a crash course of what you'll need to do to get up and running fast:

1. Create a PayPal Sandbox Account if you don't already have one and be sure to populate it with a test business account and at least one test personal account.
2. Visit the PayPal SDK page and download the SDK for your version of Java.

PayPal will soon be ending its Java 1.5 support.

3. From the *lib* directory of the PayPal SDK copy the following *JAR* files to your webapp's *WEB-INF/lib* directory:
   - *paypal_platform_stubs_AP.jar*
   - *paypal_platform_base_AP.jar*
   - *httpclient-4.0.1.jar*
   - *httpcore-4.0.1.jar*
4. Copy */samples/JSP/adaptivepayments/web/config/paypal_sdk_client.properties* into your Apache Tomcat's *conf* directory.
5. Edit *paypal_sdk_client.properties* to suit your needs. The comments in the file should be sufficient, but pay special attention to the *X-PAYPAL-SECURITY-USERID*, *X-PAYPAL-SECURITY-PASSWORD*, and *X-PAYPAL-SECURITY-SIGNATURE*. The correct values for these are all available in your sandbox account under "API Credentials". If you don't see them, make sure you have created a test business account in the sandbox.
6. From the updated code.zip, you'll need to copy *wdi_gdf_with_payments.jsp*, *vrupees.jsp*, *purchase.jsp*, *payment_response.jsp*, and *constants.jsp* into your webapp's root.
7. Replace the RECEIVER_EMAIL variable in *constants.jsp* with your own PayPal business account email address.

V-Rupees for Graphs

Now that that's out of the way, you should be able to point your browser at *http://localhost:8080/myWebApp/wdi_gdf_with_payments.jsp* to see the familiar interface we built in the last article, with a couple of additions. At the top, you'll notice the available V-Rupees on display (if this is the first time the page has been loaded it will be zero) and a select box of V-Rupees for purchase. Some new logic has also been put in place to detect if there are no more available V-Rupees and stop the user from loading a graph. Go ahead and click the "Generate Graph" button to get an alert.

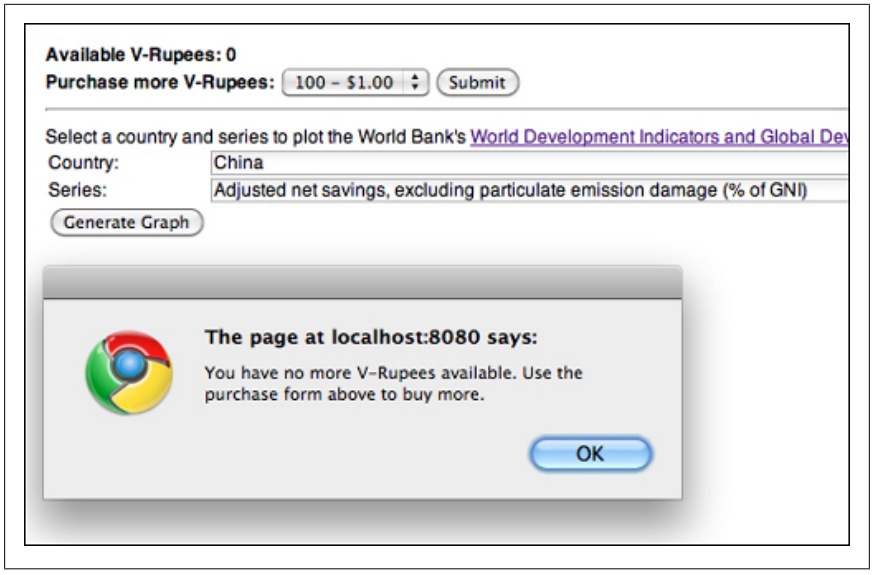

Available V-Rupees: 0
Purchase more V-Rupees:
100 – $1.00
Submit
Select a country and series to plot the World Bank's World Development Indicators and Global Dev
Country:
China
Series:
Adjusted net savings, excluding particulate emission damage (% of GNI)
Generate Graph
The page at localhost:8080 says:
You have no more V-Rupees available. Use the purchase form above to buy more.
OK

Figure 5-2. No V-Rupees, no graph!

The first thing to make note of is that the JavaScript that executes when the "Generate Graph" button is clicked uses a bit of Ajax goodness to ask the server for the number of V-Rupees a user has. If there are more than zero V-Rupees available, the graph is generated and one V-Rupee is debited from their account, or else they get the alert shown above. Dojo's `xhrGet` and `xhrPost` methods are used to perform the asynchronous calls to some server logic (*vrupees.jsp*) for updating the V-Rupees as seen below:

Example 5-1. Using Ajax to get and set V-Rupees

```
// Query for available V-Rupees and call loadSeriesForCountry if more
// than zero
function generateGraph() {
    var dfd = dojo.xhrGet({
        url: "vrupees.jsp",
        handleAs: "json",
        load: function(result) {
            if(result.vrupees <= 0) {
                alert("You have no more V-Rupees available. Use the
purchase form above to buy more.");
                return;
            }
            loadSeriesForCountry();
        },
        error: function(error) {
            alert(error);
        }
    });
}

/** Inside of loadSeriesForCountry method **/
// Add an event listener for the "ready" event so we know when the graph
// has been rendered and is ready for use. At this time we need to reduce
// the number of available V-Rupees
google.visualization.events.addListener(vis, 'ready', function(event) {
    // POST a value of -1 to the vrupees.jsp page to reduce the available
    // number of V-Rupees in the session
    dojo.xhrPost({
        url: "vrupees.jsp",
        handleAs: "json",
        content: {
            amount: -1
        },
        load: function(result) {
            updateVRupees(result.vrupees);
        },
        error: function(error) {
            alert(error);
        }
    });

});
```

You can probably deduce from the above that Ajax calls *vrupees.jsp* to both get and set the number of V-Rupees a user has. Looking at *vrupees.jsp* you'll notice that the amount of V-Rupees is actually stored in a user's session data. This is not that secure (OK, it's not secure at all in the current implementation), but it could easily be adapted to use a database instead—which will be left up to the curious reader as an exercise.

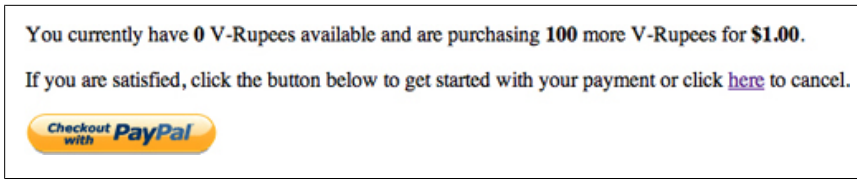

You currently have 0 V-Rupees available and are purchasing 100 more V-Rupees for $1.00.
If you are satisfied, click the button below to get started with your payment or click here to cancel.
Checkout with PayPal

Figure 5-3. The Purchase Page

Generating a Pay Key with the AdaptivePayments API

Let's go ahead and purchase some V-Rupees by selecting the number you want from the select box and clicking the Submit button. You'll be taken to the *purchase.jsp* page where the Embedded Payments fun begins.

A few important things happen behind the scenes when this page loads. First, we build up a PayRequest object with some required ingredients: the receiver of the payment with the type, amount, and receiver (that would be you), the URLs PayPal will call on a successful or canceled payment, and a RequestEnvelope. Next, we instantiate an AdaptivePayments object that uses the properties file we copied and modified earlier. Finally, we execute the AdaptivePayments' *"pay"* method with the PayRequest we've created and pull out the pay key from the response. The pay key is used later when initializing the EmbeddedPayments flow.

Example 5-2. Use the AdaptivePayments API to generate a pay key

```
<%
    // Get the number of V-Rupees from the session. This not exactly the most
    // secure way of keeping track of virtual currency and would ideally be
    // handled in a database of some sort. Of course, authentication would be
    // necessary as well, but all of that is beyond the scope of the article
    // this code is written for.
    Integer availableVRupees = (Integer)session.getAttribute("availab
leVRupees");
    if(availableVRupees == null) {
        availableVRupees = 0;
    }

    String payKey = "";
    Integer numVRupees = null;
    BigDecimal paymentAmount = null;
    Boolean stop = false;

    PayRequest payRequest = null;

    try {
        // Try to convert the POST parameter to an integer
        numVRupees = Integer.parseInt(request.getParameter("vrupees"));
```

```
        // Must be a minimum of 100
        if(numVRupees < 100) {
            stop = true;
            response.sendRedirect(WDI_REDIRECT_URL + "?error=Must buy
 a minimum of 100 V-Rupees.");
        }

        // Calculate the cost for later
        paymentAmount = new BigDecimal(numVRupees/100).setScale(2,
BigDecimal.ROUND_HALF_EVEN);

        // Build up the base of our return URLs. The need to be fully qualified
        // URLs so PayPal can properly redirect back to us.
        String url = "http://"
            + request.getServerName()
            + ":"
            + request.getServerPort()
            + request.getContextPath()
            + "/payment_response.jsp";

        // Our return URL should include the type of response it is (t=success)
        // and the number of VRupees purchased so we can update the user's available
        // rupees after the purchase. The payKey is also returned so we can request
        // the payment details to verify the purchase later.
        String returnUrl = url + "?type=success&payKey=${payKey}&vrupees="
+ numVRupees.toString();
        String cancelUrl = url + "?type=canceled&payKey=${payKey}";

        // Initialize and setup our PayRequest
        payRequest = new PayRequest();
        payRequest.setActionType("PAY");
        payRequest.setCurrencyCode("USD");

        // Use the default RequestEnvelope
        payRequest.setRequestEnvelope(new RequestEnvelope());

        // Add our return and cancel URLs
        payRequest.setReturnUrl(returnUrl);
        payRequest.setCancelUrl(cancelUrl);

        // We are not using the tracking id, but it is useful to have an id
        // in your database for when the purchase begins that allows us to
        // map back to the details of the purchase. This ID must always be
        // unique and PayPal enforces no duplicates.
        //payRequest.setTrackingId(UNIQUE_TRACKING_ID);

        // Build up a list of people to receive the money. In our case
        // we have one receiver (our business account hopefully ;)
        // NOTE: The DIGITALGOODS payment type is for embedded payments
        ReceiverList receiverList = new ReceiverList();
        Receiver rec1 = new Receiver();
            rec1.setAmount(paymentAmount);
            rec1.setEmail(RECEIVER_EMAIL);
```

```
            rec1.setPaymentType("DIGITALGOODS");
            receiverList.getReceiver().add(rec1);
        payRequest.setReceiverList(receiverList);

        // Create an instance of the AdaptivePayments object
        AdaptivePayments ap = new AdaptivePayments(PAYPAL_PROPERTIES_FILE);

        // Issue the PayRequest and receive a PayResponse in return.
        // This will contain the payKey used in the Embedded Payments
        // flow down below (in the form.)
        PayResponse payResp = ap.pay(payRequest);
        payKey = payResp.getPayKey();
    }
    catch(NumberFormatException e) {
        stop = true;
        response.sendRedirect(WDI_REDIRECT_URL + "?error=" +
e.toString());
    }
    // PayPal exception
    catch(PPFaultMessage e) {
        stop = true;

        /*for(com.paypal.svcs.types.common.ErrorData ed :
e.getFaultInfo().getError()) {
            out.print(ed.toString() + "<br>");
        }*/
        //out.print(e.getFaultInfo().getError().size());
        for(com.paypal.svcs.types.common.ErrorData ed: e.getFaultInfo
().getError()) {
            out.print("DOMAIN: " + ed.getDomain() + "<br>");
            out.print("ERROR ID: " + ed.getErrorId() + "<br>");
            out.print("EXCEPTION ID: " + ed.getExceptionId() + "<br>");
            out.print("MESSAGE: " + ed.getMessage() + "<br>");
            out.print("SUBDOMAIN: " + ed.getSubdomain() + "<br>");
        }
    }
    // Unhandled exception...grab the stack and print directly
    // to the page
    catch(Exception e) {
        stop = true;

        java.io.StringWriter sw = new java.io.StringWriter();
        java.io.PrintWriter  pw = new java.io.PrintWriter(sw);
        e.printStackTrace(pw);
        out.print(sw.toString().replace("\n", "<br>"));
    }

    // An error occurred, don't continue processing the rest of the JSP
    if(stop) {
        return;
    }
%>
```

Kickstart the Embedded Payments Flow

After getting the pay key, all that's left is to include some PayPal-provided JavaScript that controls the Embedded Payments experience, set up an HTML form with some required parameters, and initialize the flow.

The Form is pretty simple and requires only a few things.

- The Form's action should point to either the sandbox or live PayPal AdaptivePayments endpoint. In the supplied code, we are using the sandbox for testing, but once all testing is complete it will need to be pointed at *"https://www.paypal.com/..."*.
- The target should be set to "PPDGFrame" so the EmbeddedPayments flow can be connected properly.
- A hidden field *expType* must be set to "light" (which is currently the only allowed value.)
- A second hidden field *payKey* should contain the pay key generated above from the AdaptivePayments pay method.
- A PayPal designated payment button that, when clicked, will start the flow.

Finally, to initialize the flow, you'll create a one-line JavaScript call to instantiate the PayPal-provided DGFlow object with the identifier of the form button that will begin the flow.

Example 5-3. The HTML form that starts the EmbeddedPayment flow

```
<html>
    <head>
        <title>Purchase V-Rupees</title>

        <!-- PayPal Embedded Payments JS -->
        <script type="text/javascript"
src="https://www.paypalobjects.com/js/external/dg.js"></script>
    </head>

    <body class="tundra" style="padding:20px">
        <div>
            <div>
                You currently have <strong><%= availableVRupees %></strong>
V-Rupees available and are purchasing <strong><%= numVRupees %></strong>
more V-Rupees for <strong>$<%= paymentAmount %></strong>.
            </div>
            <p>
                If you are satisfied, click the button below to get started with
your payment or click <a href="<%= WDI_REDIRECT_URL %>">here</a> to cancel.
            </p>
```

```
            <form action= "https://www.sandbox.paypal.com/webapps/adaptivepayment/
flow/pay" target="PPDGFrame">
                <input id="type" type="hidden" name="expType" value="light">
                <input id="paykey" type="hidden" name="payKey"
value="<%= payKey %>">
                <input id="submitBtn" type="image" name="submit"
src="https://www.paypalobjects.com/en_US/i/btn/btn_xpressCheckout2.gif">
            </form>
        </div>
        <script type="text/javascript">
            var dgFlow = new PAYPAL.apps.DGFlow({ trigger: 'submitBtn' });
        </script>
    </body>
</html>
```

When the designated button to trigger the flow is clicked, PayPal's JavaScript kicks in and creates a lightbox. There are a couple of different scenarios that can occur here. If the user has not logged in via the Embedded Payment flow before (determined by a cookie set by PayPal) they will be presented with a simple Login button that will pop open a new window to finish the login and payment process.

PayPal
Log in to your PayPal account to complete this purchase.
Log In
New to PayPal?
Sign up for a PayPal account and complete this purchase now.
Or check out with a debit or credit card – no PayPal account required.
VISA
MasterCard
DISCOVER

Figure 5-4. EmbeddedPayment Lightbox - Login

When the popup window is presented for login, the user will be asked to enter their email and password for login. For testing you will want to set up a personal account in your PayPal sandbox and use those credentials for logging in.

You must also be logged into your PayPal sandbox account before you can use your test and business accounts.

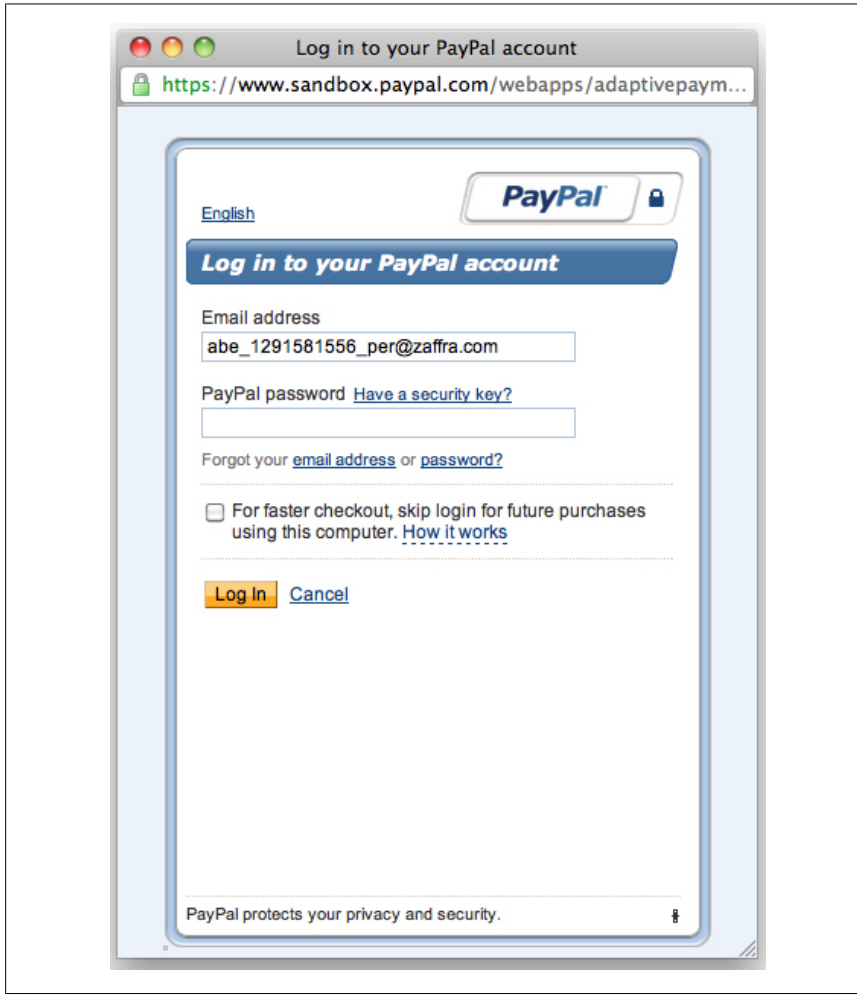

Log in to your PayPal account
https://www.sandbox.paypal.com/webapps/adaptivepaym...
English
PayPal
Log in to your PayPal account
Email address
abe_1291581556_per@zaffra.com
PayPal password Have a security key?
Forgot your email address or password?
For faster checkout, skip login for future purchases using this computer. How it works
Log In
Cancel
PayPal protects your privacy and security.

Figure 5-5. EmbeddedPayment Window - Login

When a user logs in through the popup window, they are presented with the option to "skip login for future purchases" by toggling a checkbox (Figure 5-5). This will set a cookie for the user and, until they log out, the embedded payment flow will be totally enclosed in the lightbox instead of popping up a window. The resulting screen for finalizing the payment process is the exact same in both the popup and lightbox versions, as seen in Figures 5-6 and 5-7.

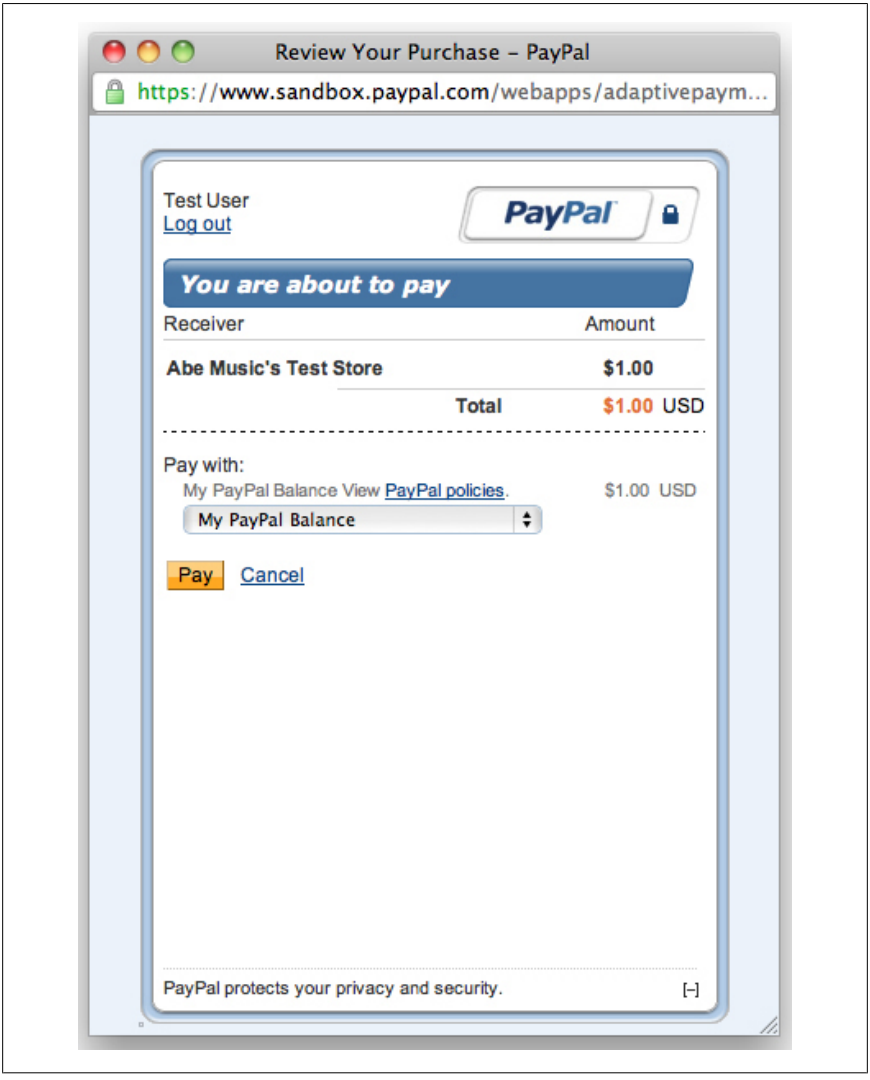


Figure 5-6. EmbeddedPayment Window - Payment

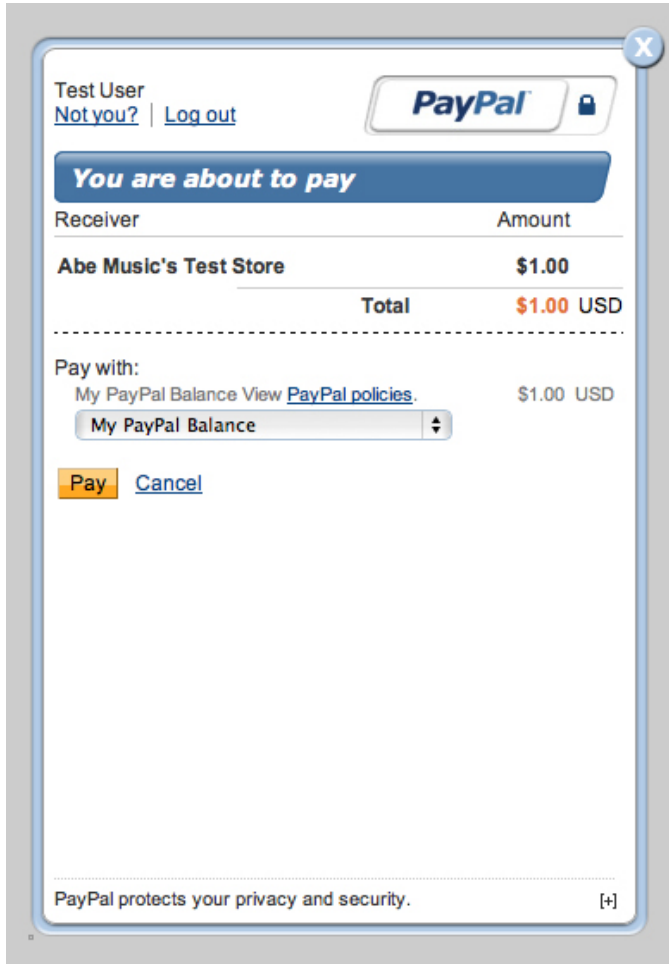


Figure 5-7. EmbeddedPayment Lightbox - Payment

Handling Payment Responses

When the PayRequest was constructed for use in the AdaptivePayments *pay* method there were two URLs set for handling both a successful and canceled payment. In the provided code, the same URL is used, but we augment the URL with some parameters set in the GET string. The first ("type") distinguishes between a success and canceled payment, and the second ("payKey") gives us the pay key for the canceled or successful payment. The pay key will

be used to construct a PaymentDetailsRequest to confirm that the payment actually went through and ended in a status of "COMPLETED". In the case of a successful payment, we also ask that PayPal return to us the number of V-Rupees purchased so we can update the session variable. Given that information, it's pretty simple to follow the logic in *payment_response.jsp* that PayPal will call when the payment process has been completed.

Example 5-4. Checking the payment with a PaymentDetailsRequest

```
/** ...SNIP... **/
// Initialize and setup a PaymentDetailsRequest
PaymentDetailsRequest pdRequest = new PaymentDetailsRequest();
pdRequest.setRequestEnvelope(new RequestEnvelope());
pdRequest.setPayKey(payKey);

// We are using the adaptive payments API
AdaptivePayments ap = new AdaptivePayments(PAYPAL_PROPERTIES_FILE);

// Send the payment details request and get a
// PaymentDetailsResponse back
PaymentDetailsResponse pdResponse = ap.paymentDetails(pdRequest);

// Here we can get the unique tracking id to look up our details about
// the purchase. I'm leaving this in only to demonstrate how it works,
// but for our purposes we aren't going to use it. Instead we use the
// GET parameter passed in to get the number of VRupees purchased.
//Integer trackingId = Integer.parseInt(pdResponse.getTrackingId());


// if the status of the payment is COMPLETED we can update the session
if(pdResponse.getStatus().equals("COMPLETED")) {
    session.setAttribute(VRUPEES_SESSION_KEY, (purchasedVRupees + ava
ilableVRupees));
}
/** ...SNIP... **/
```

After validation of the payment details, the only thing left to do is properly dispose of the DGFlow object and close the popup window if necessary. The user is also redirected back to the *wdi_gdf_with_payments.jsp* page where they can start spending their V-Rupees!

Example 5-5. Destroy the DGFlow and popup window

```
// Custom logic for the purchase would go here (e.g, updating the
// page with a custom thank you banner.) We're going to keep it simple
// and just redirect to the graph page so they can spend their new
// VRupees!

// Since we are redirecting, we don't actually need to close the light
// box, but if you wanted to stay on the same page you should call the
// dgFlow.closeFlow() method so the embedded payments flow is properly
// terminated.
```

```
// When using a popup window to purchase
if(top && top.opener && top.opener.top) {
    // Close the DGFlow
    //top.opener.top.dgFlow.closeFlow();

    // Redirect the page
    top.opener.top.location = "<%= WDI_REDIRECT_URL %>";

// When using the lightbox to purchase (i.e., when the user has already
// logged in with the remember me checkbox toggled
} else if(top && top.dgFlow) {
    // Close the DGFlow
    //top.dgFlow.closeFlow();

    // Redirect the page
    top.location = "<%= WDI_REDIRECT_URL %>";
}

// Close the popup window if necessary. Not sure why PayPal is
// not doing this for us when they do handle the closing of the
// popup window pre-login.
if(top && top.name == "_popupFlow") {
    setTimeout(top.window.close, 1000);
}
```

Wrapping Up

At this point everything should be cleaned up and the user will be sitting comfortably back at the graph-generation page. If the payment process was finalized, the user will have some V-Rupees to spend and the Generate Graph button should allow graphs to be rendered. This concludes our series on mining, visualizing, and monetizing World Bank's WDI data set. A next step you might consider for improving the code is to properly secure the application by tying a user to an account in a database instead of a session. Thanks for reading and, if you have any questions, feel free to leave a comment!

CHAPTER 6

Build an MLM Engine with Neo4j and MassPay, Part 1

John Wheeler

What Is Multi-Level Marketing?

Multi-level marketing (MLM), or *network selling*, is a strategy for maximizing sales through a network of *distributors*. Distributors are paid commissions for personal sales and the sales of others they recruit. A hierarchy forms in which new recruits are placed under the distributors who recruit them, and commissions are paid several levels up. Consequently, the earlier a person joins, the larger his or her *downline* and potential commission will be.

MLM strategies come in different shapes and sizes and vary in sophistication. We implement one called the *unilevel plan*, which is easy to understand. Basically, distributors recruit as many as they can into their *frontlines*, and frontlines recruit as many as they can into second-lines, and so on. Further examples of MLM strategies are the *binary plan*, in which frontlines can have no more than two distributors apiece, and the *forced matrix plan*, which stipulates a maximum downline width and depth, for example 3x9. These two plans usually involve membership fees paid upline as new recruits *spillover* into leaf positions.

Figure 6-1 shows a visual depiction of different MLM compensation plans.

In this two-part article, we build a unilevel MLM engine using a persistent graph library called Neo4j and PayPal's MassPay API. Part I focuses on MLM abstractions and Neo4j, and Part II focuses on MassPay.

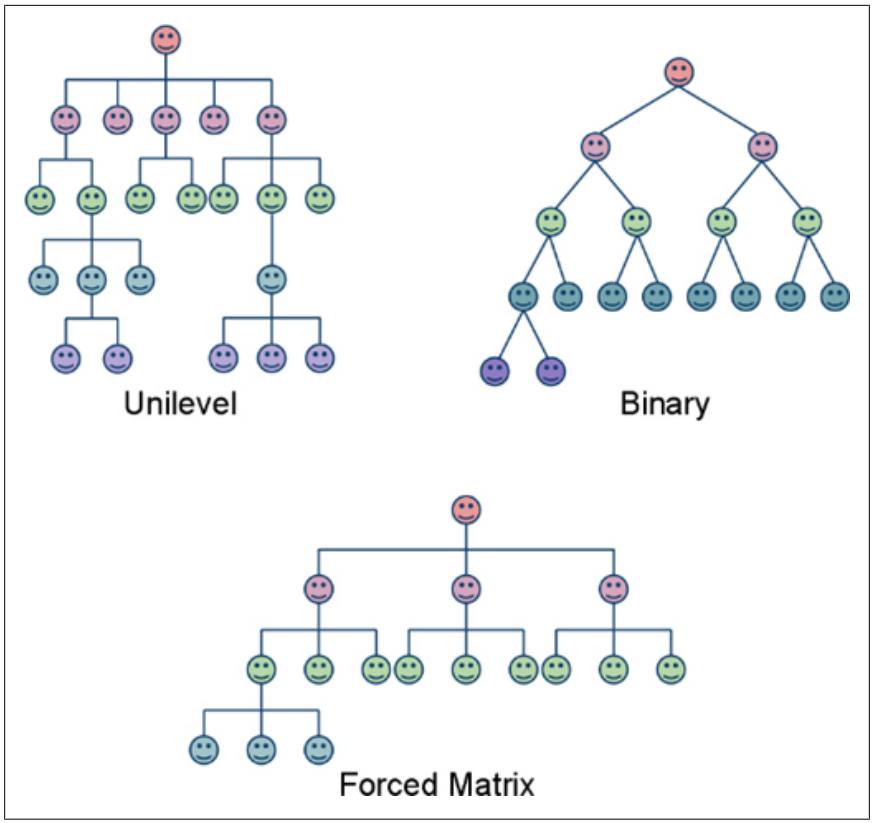


Figure 6-1. Three different MLM compensation plans

MLMs Are Graphs

A *graph* is a data structure that connects things together. Whatever is connected is called a *node*, and the connection itself is called an *edge*. Both nodes and edges might have *attributes*, which are pieces of information. A good example of a graph with node and edge attributes is a street map. With this example, locations are nodes, geocoordinates are node attributes, routes are edges, and route distances are edge attributes. The *depth* of a graph is how many levels it has, and a subset of connected nodes is called a *subgraph*. Figure 6-2 shows the anatomy of a graph.

Graphs are important because many things are graphs. Social networks, power grids, and indeed MLMs are all graphs. When programming with Neo4j, it's best to build abstractions over nodes and edges instead of working with them directly. This way, clients work with our classes more naturally. In the MLM abstraction we build, distributors are nodes that recruit, or *sponsor*, other

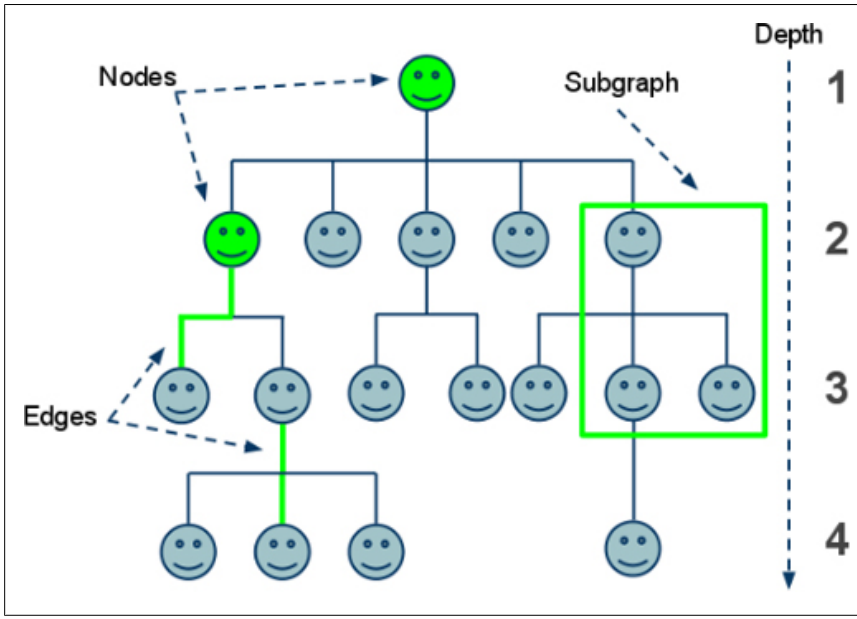


Figure 6-2. Anatomy of a graph

distributors, and these sponsorships are edges. In Neo4j, edges are called *relationships*, so we use that term from now on. Figure 6-3 shows how the MLM abstraction we build maps to a graph. Each MLM class wraps the construct it maps to as indicated by the green lines.

Now that we know how MLMs and graphs work, let's get to building our engine.

The Design and Implementation of an MLM Engine

In this section, an MLM engine is built, starting with its distributors. Then the `Distributors` are given the ability to sponsor one another through `Sponsorships`. We don't want Neo4j `Nodes` and `Relationships` visible outside of our classes, so we prevent access with the `protected` modifier. That way, clients can work naturally with our objects without any prior knowledge of graphs. Finally, in this section, we build a handy `SalesNetwork` factory for creating `Distributors` more easily.

For demonstration purposes, `Distributor` has just two attributes: a `sales double` for computing commissions and an `email` PayPal pays to. They aren't kept on `Distributor` but on its underlying node, and access is delegated. Create a class called `Distributor` and put in the following code:

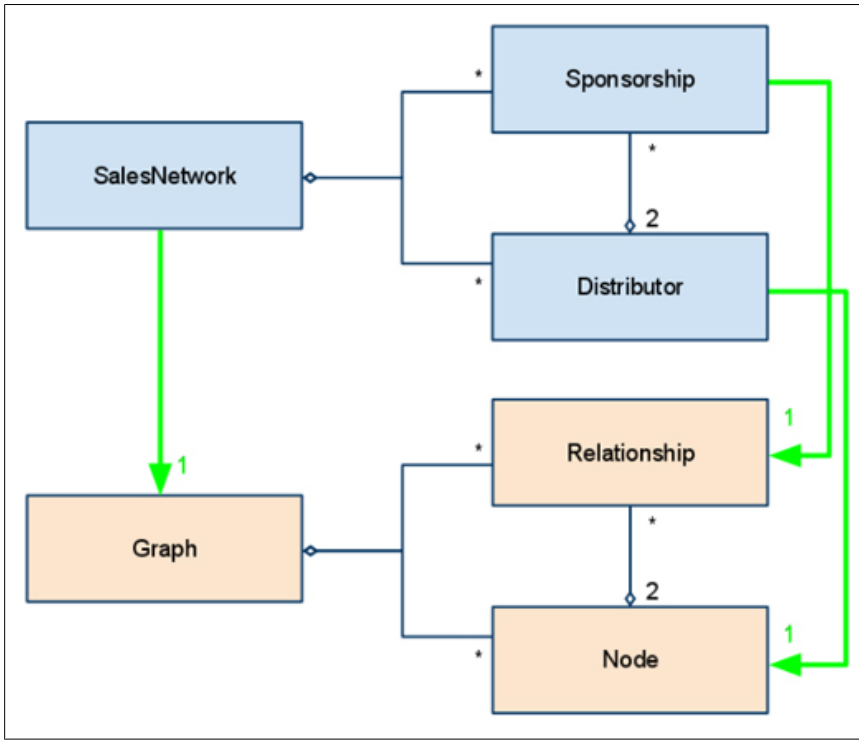


Figure 6-3. How the MLM domain model fits over a graph

```
package info.johnwheeler.samples.domain;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.HashSet;
import java.util.Set;

import org.neo4j.graphdb.Direction;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.RelationshipType;
import org.neo4j.graphdb.ReturnableEvaluator;
import org.neo4j.graphdb.StopEvaluator;
import org.neo4j.graphdb.Traverser;
import org.neo4j.graphdb.Traverser.Order;

public class Distributor {
    private Node underlyingNode;
    protected int level;

    protected Distributor(Node node) {
        underlyingNode = node;
    }
```

```
    protected Distributor(Node node, String email, double sales) {
        this(node);
        setEmail(email);
        setSales(BigDecimal.valueOf(sales));
    }

    public long getId() {
        return underlyingNode.getId();
    }

    protected void setLevel(int level) {
        this.level = level;
    }

    protected int getLevel() {
        return level;
    }

    public void setEmail(String email) {
        underlyingNode.setProperty("email", email);
    }

    public String getEmail() {
        return (String) underlyingNode.getProperty("email");
    }

    public void setSales(BigDecimal sales) {
        underlyingNode.setProperty("sales", sales.doubleValue());
    }

    public BigDecimal getSales() {
        return BigDecimal.valueOf((Double) underlyingNode.getProperty
("sales"));
    }

    public boolean equals(Object other) {
        Distributor that = (Distributor) other;
        return this.underlyingNode.equals(that.underlyingNode);
    }

    public int hashCode() {
        return underlyingNode.hashCode();
    }

    protected Node getUnderlyingNode() {
        return underlyingNode;
    }
}
```

As previously mentioned, `Distributor` delegates to `Node` for most of its affairs. `Distributor` has two `protected` constructors, which only classes in the same package and subclasses have access to. A `SalesNetwork` factory uses the

constructors later to dole out `Distributor`s and hide `Node` from clients. We want to store the `level` variable outside `underlyingNode`. It's an implementation detail of commission calculations later on and not part of the abstraction; therefore, it's also `protected`. Lastly, note how `equals` and `hashCode` also delegate to `Node` since it manages state. Before moving on, it's worth mentioning that the Neo4j wiki shows another approach to encapsulation that uses interfaces.

Now, we move forward and enable `Distributor`s to sponsor one another. `Relationship`s in a Neo4j graph are typed to differentiate themselves from one another. For example, in a social network like Facebook, people "friend" and block one another. In Neo4j, both types of connection would be realized through `RelationshipType`s. `RelationshipType` is an `enum` that's extended in order to define more `RelationshipType`s. Add the following code to the `Distributor` class.

```
public static enum RelTypes implements RelationshipType {
    SPONSOR, DISTRIBUTORS
}
```

Now we can use the `enum` to create `Relationship`s between `Node`s or more concretely: `Sponsorship`s between `Distributor`s. Create a file called *Sponsorship.java* with the following code.

```
package info.johnwheeler.samples.domain;

import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.Relationship;

public class Sponsorship {

    private Relationship underlyingRel;

    protected Sponsorship(Distributor sponsor, Distributor recruit) {
        Node n0 = sponsor.getUnderlyingNode();
        Node n1 = recruit.getUnderlyingNode();
        underlyingRel = n0.createRelationshipTo(n1,
Distributor.RelTypes.SPONSOR);
    }

    protected Relationship getUnderlyingRelationship() {
        return underlyingRel;
    }

    public boolean equals(Object other) {
        Sponsorship that = (Sponsorship) other;
        return this.underlyingRel.equals(that.underlyingRel);
    }

    public int hashCode() {
        return underlyingRel.hashCode();
```

```
    }
}
```

`Sponsorship` follows the same delegation pattern as `Distributor`. Moreover, a `protected` constructor brings two `Distributor`s together. Although it doesn't cause the abstraction to leak, we want to prevent API clients from using it and instead rely on `Distributor`s to do the sponsoring. It's a more object-oriented approach. Add the method below to `Distributor`.

```
public Sponsorship sponsor(Distributor recruit) {
    return new Sponsorship(this, recruit);
}
```

`Distributor` and `Sponsorship` are taken care of for now, so the `SalesNetwork` factory is next. It wraps the graph itself and creates `Distributor`s. Earlier, we mentioned Neo4j persists automatically. That's a true statement with a small detail left out: graph-modifying operations must execute in transactions. `SalesNetwork` helps with that too. Create a file named *SalesFactory.java* with the code below.

```
package info.johnwheeler.samples.domain;

import info.johnwheeler.samples.domain.Distributor.RelTypes;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

import org.neo4j.graphdb.Direction;
import org.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.Relationship;
import org.neo4j.graphdb.ReturnableEvaluator;
import org.neo4j.graphdb.StopEvaluator;
import org.neo4j.graphdb.Transaction;
import org.neo4j.graphdb.Traverser;
import org.neo4j.graphdb.Traverser.Order;
import org.neo4j.kernel.EmbeddedGraphDatabase;

public class SalesNetwork {

    protected GraphDatabaseService db;
    protected Transaction tx;

    public static SalesNetwork open() {
        return new SalesNetwork();
    }

    protected SalesNetwork() {
        db = new EmbeddedGraphDatabase("var/db");
        tx = db.beginTx();
    }
```

```
    public void close(boolean graceful) {
        if (graceful) {
            tx.success();
        } else {
            tx.failure();
        }

        tx.finish();
        db.shutdown();
    }
}
```

So far, the class above is just a stub with lifecycle methods. `GraphDatabaseService` is our main interface into Neo4j. It creates new `Node`s and looks up existing ones by ID. Additionally, `GraphDatabaseService` is constructed with a path pointing to where the database is created if it doesn't exist. Neo4j automatically handles persistence, so nodes and edges are stored in a database as they're created. The lifecycle methods, `open` and `close`, start and shut down the database and also manage transactions. Each Neo4j graph modifying operation must take place in one. The idiom is to call `beginTx`, perform some graph operations, and either call `success` or `failure` before `finish` depending on whether or not an exception is thrown.

Neo4j databases are schemaless, but there's a special node called the *reference node* for attaching graph roots to in order to get a handle on them later. The next method creates the first `Distributor` in the system and attaches it to the reference node. All subsequent `Distributor`s are descendants of this one. Add the method to `SalesNetwork`

```
public Distributor manager(String name, double sales) {
    Distributor manager = new Distributor(db.createNode(),
name, sales);
    Node managerNode = manager.getUnderlyingNode();
    Node refNode = db.getReferenceNode();

    refNode.createRelationshipTo(managerNode,
Distributor.RelTypes.DISTRIBUTORS);
    return manager;
}
```

Common parlance in the MLM industry is to call the first distributor a *manager*. In the method above, a `Distributor` instance is created before placing a handle on its underlying node. Next, the aforementioned reference node is obtained, and a relationship is formed between the two with `createRelationshipTo`. Since `refNode` is making the call, the relationship is `directed` towards `managerNode`. Figure 6-4 shows different graphs connecting to the reference node, so they can be retrieved in their entirety later. The reference node is the green one at the top.

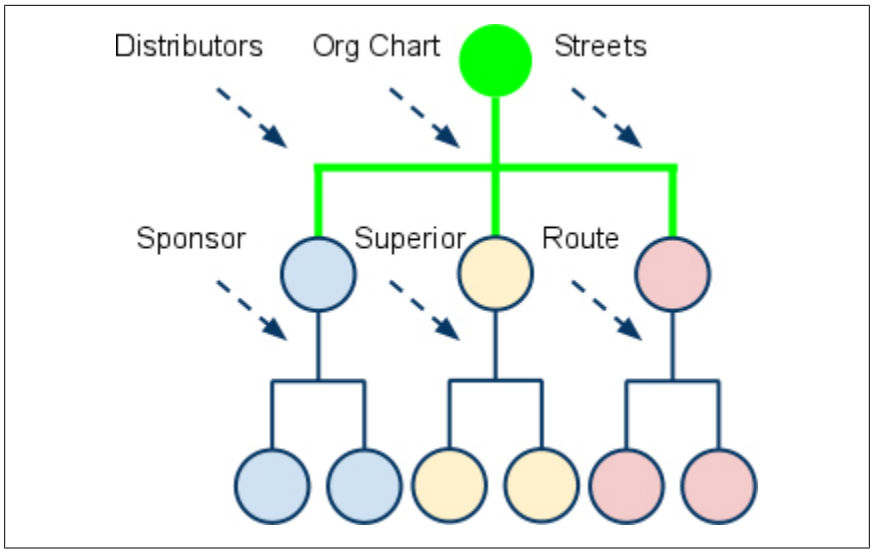


Figure 6-4. Nodes form classifying connections with the reference node to build a schema

Now let's write a run-of-the-mill factory method on `SalesNetwork` for recruiting additional distributors. Add the method below.

```
public Distributor recruit(String name, double sales) {
    Node refNode = db.getReferenceNode();
    Iterable<Relationship> i =
refNode.getRelationships(Distributor.RelTypes.DISTRIBUTORS);

    if (!i.iterator().hasNext()) {
        throw new IllegalStateException("must call manager first");
    }

    Distributor distributor = new Distributor(db.createNode(),
name, sales);
    return distributor;
}
```

In order to recruit a new `Distributor`, first the reference node is checked for the presense of a manager. If one doesn't exist, an exception prompts developers to create it before proceeding. Then, the graph is asked to create a `Node`, which is passed into the new `Distributor` instance returned to clients.

Now that we have these fairly simple structures fleshed-out, we turn our attention to computing commissions by searching through, or *traversing*, the graph.

Traversing the Graph to Compute Commissions

Traversing a graph means visiting its nodes or a subset of its nodes in a particular order. A *breadth-first* traversal scans across levels in a graph's heirarchy before going deeper into them. A *depth-first* search goes deep before going across. Figure 6-5 shows the path each traversal takes in a sample graph.

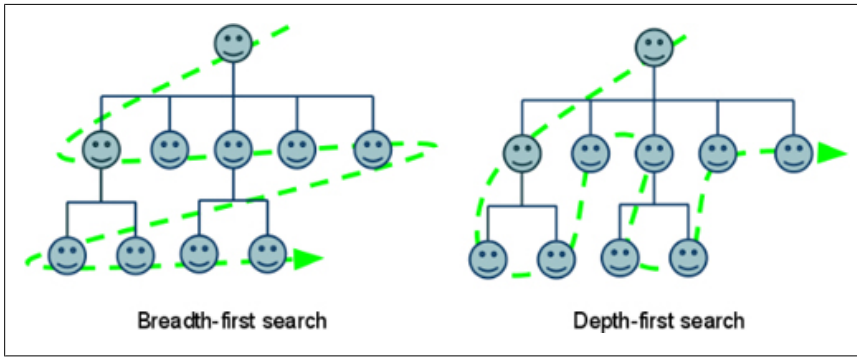


Figure 6-5. Breadth-first vs. depth-first graph traversing

Commissions come from a `Distributor`'s downline. Total sales are calculated at each level with a breadth-first search and matched against a percentage table built in the step after this one. Add the method below to `Distributor` to get a hold of the downline.

```
public Set<Distributor> downline() {
    Set<Distributor> downline = new HashSet<Distributor>();

    Traverser nodes = underlyingNode.traverse(Order.BREADTH_FIRST,
StopEvaluator.END_OF_GRAPH,
            ReturnableEvaluator.ALL_BUT_START_NODE, RelTypes.SPONSOR,
Direction.OUTGOING);

    for (Node node : nodes) {
        Distributor distributor = new Distributor(node);
        distributor.setLevel(nodes.currentPosition().depth());

        downline.add(distributor);
    }

    return downline;
}
```

`downline` returns results of the breadth-first search in a `HashSet`. The second line of code shows Neo4j's traversal API. The meat of it is in the method arguments, so let's go over those in turn.

There are two ordering options: `BREADTH_FIRST` or `DEPTH_FIRST`. We already discussed what those mean.

`StopEvaluator` is an interface implemented to stop traversal for whatever reason. It has one method: `isStopNode`. Two out-of-the-box implementations are supplied as constants on the interface. The one we use, `END_OF_GRAPH`, traverses to the end of the graph. The other one, `DEPTH_ONE`, returns a single level of nodes; that would mean the frontline in our case.

`ReturnableEvaluator` is a hook for determining whether a traversed `Node` is part of the returned set. Two out-of-the-box implementations exist for it too. `ALL_BUT_START_NODE` returns the downline, which does not include the parent `underlyingNode`. If we want the parent, we use `ALL`.

The next argument is the `RelationshipType` to return, which is `SPONSOR` in our case. The reference node along with `DISTRIBUTOR` returns the complete graph.

The final argument is the direction to traverse relative to `underlyingNode`. `OUTGOING` returns the downline and `INCOMING` the upline. `BOTH` returns everything. A record of each `Distributor`'s depth is kept in the `level` variable as the graph is traversed. `level` is matched directly against a commission table that we build next.

Create a class called `Commissions` with the code below.

```
package info.johnwheeler.samples.domain;

import java.math.BigDecimal;

public final class Commissions {

    private static Commissions soleInstance;
    private double[] percentages;

    public static void setLevels(double... percentages) {
        if (soleInstance != null) {
            throw new IllegalStateException
("setLevels already called");
        }

        soleInstance = new Commissions(percentages);
    }

    public static Commissions getInstance() {
        if (soleInstance == null) {
            throw new IllegalStateException("must setLevels first");
        }

        return soleInstance;
    }

    private Commissions(double... percentages) {
```

```
            this.percentages = percentages;
        }
    }
```

`Commissions` is a singleton. Singletons are single instances of a class that exist from the time they're created until the virtual machine quits. The design pattern fits because there's one commission table, and this saves us the trouble of passing it around from method to method. The mechanics of singletons have to do with their `private` constructors and `static` self-references. We don't have time to do the pattern justice in this article, but there's literature all over the Internet. Moving on with the code, percentages are set externally with `setLevels`.

The code above doesn't do anything until we put in the method below for matching levels to percentages.

```
public BigDecimal atLevel(int i) {
    int levelOffset = i - 1;

    if (levelOffset >= percentages.length) {
        return BigDecimal.ZERO;
    }

    return BigDecimal.valueOf(percentages[levelOffset]);
}
```

`atLevel` matches `Distributor` levels to percentages. Figure 6-6 shows an example of commissions calculated recursively downline. Level 1 is for personal commission, level 2 is for frontline commission, and level 3 is for second-line commission.

Now we calculate downline commission, personal commission, and both together for total commission. The next `protected` helper method determines commission payouts at a particular level. Add it to `Distributor`.

```
protected BigDecimal commissionAtLevel(int level) {
    BigDecimal sales = getSales();
    BigDecimal percentage = Commissions.getInstance().atLevel(level);
    BigDecimal payout = sales.multiply(percentage).divide
(BigDecimal.valueOf(100), 2, RoundingMode.HALF_EVEN);

    return payout;
}
```

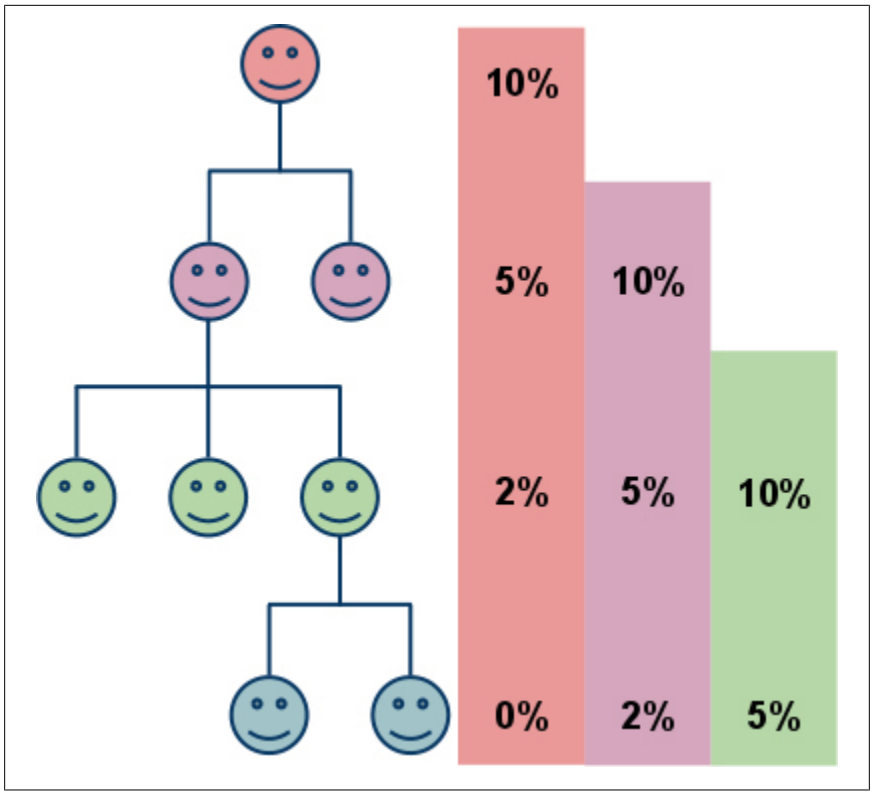


Figure 6-6. How levels match against the commission table

The routine above is simple multiplication that requires percentages as whole numbers instead of fractions. It's been done that way to make things more user-friendly (i.e., `10` instead of `0.1`). Use the routine downline by adding the next method to `Distributor` too.

```
public BigDecimal downlineCommission() {
    BigDecimal commission = BigDecimal.ZERO;

    for (Distributor distributor : downline()) {
        int nextLevel = distributor.getLevel() + 1;
        commission = commission.add(distributor.commissionAtLevel(
nextLevel));
    }

    return commission;
}
```

In the code above, we go one past the level of the current node to get the first downline member and continue doing so until the traversal results are exhausted. Personal commission is calculated at level 1.

```
public BigDecimal personalCommission() {
    return commissionAtLevel(1);
}
```

And finally, total commission is the sum of downline and personal commission.

```
public BigDecimal commission() {
    return personalCommission().add(downlineCommission());
}
```

Wrapping Things Up

That about does it for this article. At this point, play with the API in a console program to get a feel before moving on to Chapter 7. Use the `Testbed` class in the sample code to guide you.

In Part I of this article, we learned how MLMs work and about graph theory. We mapped an MLM abstraction on top of the Neo4j persistent graph API, built a commission table, and used the commission table in a series of calculations. In the next part of this article, we pay downline commissions with MassPay using the builder pattern to make it easier.

CHAPTER 7

Build an MLM Engine with Neo4j and MassPay, Part 2

John Wheeler

Introduction

In the first part of this article, we used the open source Neo4j graph library to compute Multi-Level Marketing (MLM) commissions for distributors across a sales network. We left off with a list of e-mails and associated payouts, which are now handed off to PayPal's MassPay API in this second and final installment.

MassPay can pay up to 250 people at once. In order to use it, e-mails and payouts are formatted as name-value pairs (NVP) and sent over HTTPS. MassPay is perhaps PayPal's most simple NVP API; therefore, it's a good candidate for demonstrating the creation of a class that's reusable across all the NVP APIs. The class we work on uses the builder design pattern to achieve what's called a fluent interface. The best way to explain is to show the code we're shooting for:

```
NvpApiBuilder api = new NvpApiBuilder(SERVER);

api.authenticateWith(USERNAME, PWD, SIGNATURE)
   .toCall("MassPay", "56.0")
   .withParam("L_EMAIL0", "foo@example.com")
   .withParam("L_AMT0", "15.0")
   .withParam("L_EMAIL1", "bar@example.com")
   .withParam("L_AMT1", "10.0")
   .go();
```

With a fluent interface, the elements and rules of an API are expressed in a mini-language made by chaining method calls together. The order and quantity of each call is usually predefined and important, and the goal is to make the whole thing human-readable and user friendly. This way, developers can learn an API through a simple block of code like the one shown in the example above. In the context of PayPal and its name-value pairs, we start by constructing an `api` object while assuming `SERVER` points to one of the PayPal environments (sandbox or production). It's easy to see what's going on from there: we authenticate with some more constants, declare which PayPal API to call, set up our name-value pairs, and finally execute the call. The diagram in Figure 7-1 shows the sequence of steps.

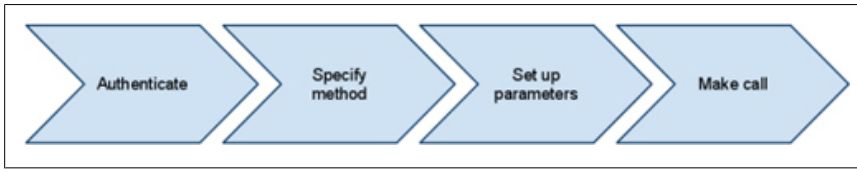


Figure 7-1. The sequence of steps involved in calling a PayPal NVP API

In this article, we construct `NvpApiBuilder` one method at a time. After that, we use it to call MassPay and pay commissions to a list of MLM distributors. Let's begin by getting some boilerplate out of the way.

Defining the Class

Open the project we left off with last time, or download the article's sample code. Stub out a class called `NvpApiBuilder` with the following:

```
package info.johnwheeler.samples.services;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.StringWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLDecoder;
import java.net.URLEncoder;
import java.util.TreeMap;

public class NvpApiBuilder {
    protected String server;
    protected StringBuilder buf;
```

```
    public NvpApiBuilder(String server) {
        this.buf = new StringBuilder();
        this.server = server;
    }
}
```

Without worrying about the imports, the class above contains a `String` for the PayPal URL and a buffer for tracking name-value pairs. The method below uses the buffer.

```
public NvpApiBuilder withParam(String name, Object val) throws
IOException {
    String n = URLEncoder.encode(name, "UTF-8");
    String v = URLEncoder.encode(val.toString(), "UTF-8");
    buf.append(n).append("=").append(v).append("&");
    return this;
}
```

Name-value pairs are sent over in an HTTP POST as URL-encoded parameters separated by ampersands. PayPal APIs define sets of parameters that it's up to us to supply values for. In addition to a few standard parameters, MassPay defines these: `L_EMAIL`*n* and `L_AMT`*n*. The former holds payee e-mails, and the latter holds payment amounts. The *n* at the end of each parameter signifies a zero-based index that's incremented for up to 250 payees. The end of the method shows the tell-tale mark of the builder pattern: returning `this`, so other methods can be chained on in the same Java statement.

Now let's flesh out the rest of the fluent interface starting with authentication and API resolution.

Authentication and API Resolution

If you don't yet have API credentials, follow these steps:

1. Log into PayPal, then click Profile under My Account.
2. Click API Access.
3. Click Request API Credentials.
4. Check Request API signature and click Agree and Submit.
5. Click Done to complete the process.

Figure 7-2 shows the API Credentials screen.

Now let's give `NvpApiBuilder` the ability to authenticate and tell PayPal which API to use. We need methods for both capabilities. The first one is below.

```
public NvpApiBuilder authenticateWith(String username, String pwd,
String signature) throws IOException {
    return withParam("USER", username).withParam("PWD", pwd).withParam
```

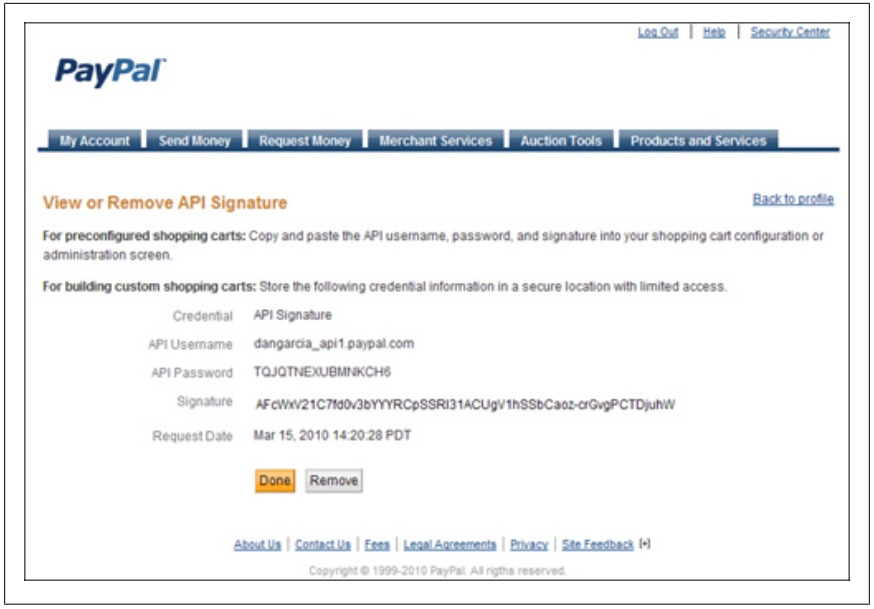

Log Out | Help | Security Center
PayPal
My Account
Send Money
Request Money
Merchant Services
Auction Tools
Products and Services
View or Remove API Signature
Back to profile
For preconfigured shopping carts: Copy and paste the API username, password, and signature into your shopping cart configuration or administration screen.
For building custom shopping carts: Store the following credential information in a secure location with limited access.
Credential
API Signature
API Username
dangarcia_api1.paypal.com
API Password
TQJQTNEXUBMNKCH6
Signature
AFcWxV21C7fd0v3bYYYRCpSSRI31ACUgV1hSSbCaoz-crGvgPCTDjuhW
Request Date
Mar 15, 2010 14:20:28 PDT
Done
Remove
About Us | Contact Us | Fees | Legal Agreements | Privacy | Site Feedback
Copyright © 1999-2010 PayPal. All rights reserved.

Figure 7-2. The API Credentials screen

```
("SIGNATURE", signature);
}
```

Above, `authenticateWith` conveniently wraps `withParam` to store security credentials in the same buffer other parameters are stored in. Here's where it puts us:

```
api.authenticateWith(USERNAME, PWD, SIGNATURE);
```

Next, `NvpApiBuilder` is told which API to use with this:

```
public NvpApiBuilder toCall(String method, String version) throws
IOException {
    return withParam("METHOD", method).withParam("VERSION", version);
}
```

`toCall` takes the name and version of the PayPal NVP API we want to use. These values differ across each NVP API and are specified in each APIs' respective documentation. In our case, we use `MassPay` for `method` and `56.0` for `version`:

```
api.authenticateWith(USERNAME, PWD, SIGNATURE).toCall
    ("MassPay", "56.0");
```

At this point, we can authenticate to PayPal, tell them which API we want, and set up API-specific name-value parameters. In the next step, we write networking code to send everything over.

Making the Call

It's a good idea to go over the networking in bite-sized chunks to make it easier to understand. In this section, we write a method to connect to PayPal, send parameters over, and process the response PayPal returns.

After a call is made to MassPay, a response is returned, which contains status fields formatted as name-value pairs. A `TreeMap` is a good structure for storing status fields because it uses keys and values and is ordered. Therefore, our method that connects to PayPal and makes API calls returns a `TreeMap`. Stub out the method with the code below.

```
public TreeMap<String, String> go() throws IOException {
}
```

While there's no shortage of Java networking APIs, we use the low-level `java.net` classes because they're easy to work with and supply all the functionality we need. The code below creates and sets up an HTTPS connection to PayPal.

```
HttpURLConnection conn = (HttpURLConnection) new URL(server).
    openConnection();
conn.setDoOutput(true);
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "text/namevalue");
conn.setRequestProperty("Content-Length", Integer.toString(params.get
Bytes().length));
```

First in the code above, an `HttpURLConnection` is made to the URL held by `server`. Then, the connection is set up to POST before a few required PayPal parameters are written as HTTP headers; however, nothing is actually being written at this point. In order to send parameters over, they're written directly to the connection's output stream. Likewise, PayPal responses are read from the connection's input stream. Examine the code below.

```
String params = buf.substring(0, buf.length()); // remove last &
conn.getOutputStream().write(params.getBytes());

InputStream in = conn.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(in));
StringWriter writer = new StringWriter();

char[] buf = new char[4096];
while (reader.read(buf) != -1)
    writer.write(buf);
```

The code above is standard for writing and reading bytes to and from streams. We push all the bytes to the connection's output stream at once and read bytes from the connection's input stream in 4 kilobytes chunks. Once we've captured the complete response, it's parsed into a `TreeMap` with the following code:

```
TreeMap<String, String> result = new TreeMap<String, String>();
String[] responseFields = writer.toString().split("&");
for (String field : responseFields) {
    int index = field.indexOf('=');
    String name = field.substring(0, index);
    String val = field.substring(index + 1, field.length());
    name = URLDecoder.decode(name, "UTF-8");
    val = URLDecoder.decode(val, "UTF-8");
    result.put(name, val);
}

return result;
```

In the code above, a `TreeMap` is constructed to store results. Then, name-value pairs are extracted into an array of `String`s by calling `String.split` using the ampersand character as a delimiter. Once the name-value pairs are all lined-up in an array, we roll through and break each pair at the "equals" character using the field name for a map key and the value portion for a map value. Everything is URL-decoded.

For convenience, the method we just worked on is shown in its entirety below.

```
public TreeMap<String, String> go() throws IOException {
    String params = buf.substring(0, buf.length()); // remove last &

    HttpURLConnection conn = (HttpURLConnection) new URL(server).
openConnection();
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "text/namevalue");
    conn.setRequestProperty("Content-Length", Integer.toString(
params.getBytes().length));
    conn.getOutputStream().write(params.getBytes());

    InputStream in = conn.getInputStream();
    BufferedReader reader = new BufferedReader(
new InputStreamReader(in));
    StringWriter writer = new StringWriter();

    char[] buf = new char[4096];
    while (reader.read(buf) != -1)
        writer.write(buf);

    TreeMap<String, String> result = new TreeMap<String, String>();
    String[] responseFields = writer.toString().split("&");
    for (String field : responseFields) {
        int index = field.indexOf('=');
        String name = field.substring(0, index);
        String val = field.substring(index + 1, field.length());
        name = URLDecoder.decode(name, "UTF-8");
        val = URLDecoder.decode(val, "UTF-8");
        result.put(name, val);
    }
```

```
        return result;
    }
```

Our `NvpApiBuilder` class is now complete. The next section shows how to use it.

Putting It All Together

With `NvpApiBuilder` complete, we're ready to invoke MassPay to payout a downline of distributors. A method we would probably attach to some payment service is defined below.

```
public static TreeMap massPay(List<Distributor> salesforce) throws
IOException {
    NvpApiBuilder api = new NvpApiBuilder(SERVER);
    api.authenticateWith(USERNAME, PWD, SIGNATURE).toCall("MassPay",
"56.0");

    int n = 0;
    for (Distributor distributor : salesforce) {
        BigDecimal commission = distributor.commission();

        if (commission.compareTo(BigDecimal.ZERO) <= 0)
            continue;

        String email = distributor.getEmail();
        api.withParam("L_EMAIL" + n, email).withParam("L_AMT" +
n, commission);
        n++;
    }

    return api.go();
}
```

In the method above, we create an instance of `NvpApiBuilder`, set up authentication, and set up which PayPal API we want to call. After that, we iterate through a downline of distributors using their fields to assemble MassPay-specific parameters. Finally, we call `go` to make the connection to PayPal and send everything over. The code attachment included with this article on X.com contains a `Testbed` class, which shows how to use the code we wrote to create a sample network and pay distributors with MassPay.

CHAPTER 8

Getting Started with PayPal on Django

A brief walkthrough on how to accept payments on your Django application, using the PayPal APIs

Peter Georgeson

Introduction

This tutorial describes how to integrate Django with PayPal by implementing a simple online store that sells downloadable content. Django is a powerful, Python-based web framework suitable for rapidly building web applications. It has an active community, is easy to use, and drives a number of prominent sites. PayPal offers an easy way to send and receive payments, has a comprehensive API, and is suitable for developers to use for online transactions.

Requirements

Paypal Accounts

If you don't already have one, create a PayPal developer account at *https://developer.paypal.com/* for testing. After creating your main developer account, you will also need to create seller and buyer test accounts. If you want to start accepting real money, you will also need to sign up for a live account at *https://www.paypal.com/*

PDT (Payment Data Transfer)

PayPal provides a number of methods for authenticating a transaction once the buyer returns to your site. This example uses the PDT method of authenticating transactions, so you need to enable PDT on your seller account. Instructions for doing this are at *https://cms.paypal.com/us/cgi-bin/?cmd=_render-content&content_ID=developer/howto_html_paymentdatatransfer*. When updating the configuration settings on your Django application, use the PDT token obtained from the above link.

Django

You need Django installed to run the application. Installation instructions for Django can be found at *http://www.djangoproject.com/download/*.

Sample Application

A fully working sample application is available from *https://github.com/supernifty/django-paypal-store-example*. I highly recommend that you download the code to follow along with the snippets shown in this tutorial.

Running the Sample Application

Configuration

To get the application up and running in your environment, some settings on the Django application need to be updated. Open the file *samplesite/settings.py* and edit the following settings:

Template path (TEMPLATE_DIRS)
: The full path to your template directory must be set.

Resource directory (RESOURCE_DIR)
: The full path to your sample resources must be set.

PayPal email address (PAYPAL_EMAIL)
: Your PayPal email address.

PayPal PDT token (PAYPAL_PDT_TOKEN)
: Your PayPal PDT token, configured earlier.

Database

To generate the basic schema with some simple data, run the command:

```
python manage.py syncdb
```

Web Server

To start the web server, issue the command:

```
python manage.py runserver
```

You can check that the test server is running by starting your web browser and browsing to *http://127.0.0.1:8000/*.

A Basic Payment Workflow

The sample application demonstrates how you might use Django and PayPal to build a store for users to purchase downloadable content, whether this is ebooks, software, images, or any other electronic, sellable file. The application follows a simple workflow, described below.

The Model

In this example, we want to track available resources, and what purchases a user has made. In a Django application, the database model is defined in *models.py*:

```
from django.db import models

class Resource( models.Model ):
  '''resources available for purchase'''
  name = models.CharField( max_length=250 )
  location = models.CharField( max_length=250 )
  price = models.DecimalField( decimal_places=2, max_digits=7 )

class Purchase( models.Model ):
  '''purchases'''
  resource = models.ForeignKey( Resource )
  purchaser = models.ForeignKey( User )
  purchased_at = models.DateTimeField(auto_now_add=True)
  tx = models.CharField( max_length=250 )
```

The *Resource* class defined above stores the available downloadable content; the *Purchase* class stores purchase details.

URL Mappings

urls.py maps a requested URL to a view defined in *views.py* using regular expressions:

```
(r'^$', 'samplesite.sampleapp.views.home' ),
(r'^download/(?P<id>\d+)/$',
    'samplesite.sampleapp.views.download' ),
```

```
# view a purchase
(r'^purchased/(?P<uid>\d+)/(?P<id>\d+)/$',
    'samplesite.sampleapp.views.purchased' ),
# purchase callback
```

In the sample application, just three mappings are required—the home page, the page for a user to request a resource, and the page displayed after a successful purchase.

Step 1—List Content Available for Purchase

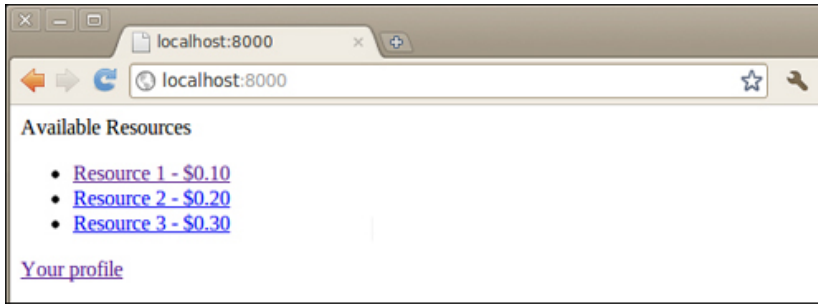
The first step in the user experience is to show the visitor what is available for purchase. This is easily achieved in *views.py*:

```
def home( request ):
  return render_to_response('home.html',
    {'list': models.Resource.objects.all() } )
```

The *home* method makes the list of all resource objects available to *home.html*. Then in the *home.html* template, we iterate over this list of available resources and display them to the user:

```
{% for item in list %}
<li><a href="/download/{{ item.id }}/">{{ item.name }} -
    ${{ item.price|floatformat:2 }}</a></li>
{% endfor %}
```

Here's what it looks like in a browser:



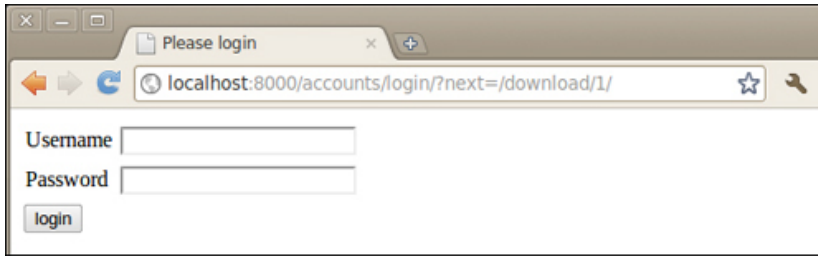
Step 2—Attempt to Access Content

If the user clicks on one of the download links, they will first be prompted to login. This enables the application to check if the user has already purchased the content, and then after purchase, to mark the content as having been purchased by this user.

Django provides a user authentication module which can be activated using an appropriate decorator in *views.py*:

```
@login_required
def download( request, id ):
```

The *@login_required* decorator simply checks to see if the HTTP request has a valid user session attached, and if not, redirects the user to a login page.



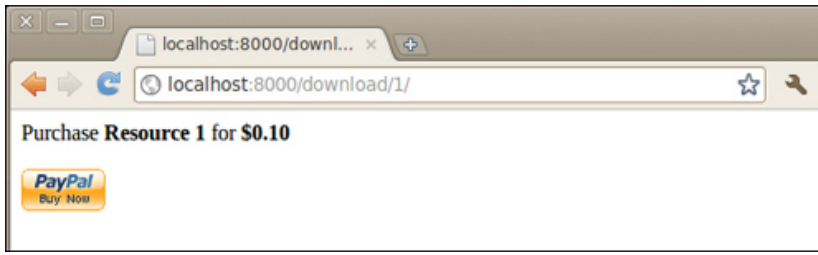
Once we have an authenticated user, the application checks to see if they have already purchased the requested content. If so, it is provided to the user. If the item has not been purchased, the user is redirected to a page prompting them to purchase the item:

```
try:
  purchased = models.Purchase.objects.get( resource=resource,
    purchaser=request.user )
  # ...download content...
except models.Purchase.DoesNotExist:
  return render_to_response('purchase.html', { 'resource': resource,
...
```

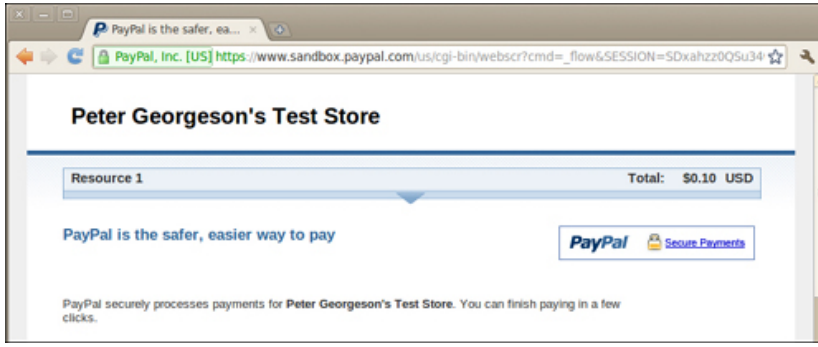
Step 3—Purchase Content

The *purchase.html* template file displays details of the content to be purchased and a form that will post this data to PayPal. Important details forwarded to PayPal include the amount to pay, what is being purchased, and a URL the user will be forwarded to once payment is complete.

```
<p>Purchase <b>{{ resource.name }}</b> for <b>${{
    resource.price|floatformat:2 }}</b></p>
<form action="{{ paypal_url }}" method="post">
  <input type="hidden" name="amount" value="{{ resource.price }}">
  <input type="hidden" name="item_name" value="{{ resource.name }}">
  <input type="hidden" name="return" value="{{ paypal_return_url }}
    /purchased/{{ user.id }}/{{ resource.id }}/">
  ...
</form>
```



Once the user clicks the "Pay Now" button, they are redirected to PayPal, and the user goes through the payment process. On completion, they are directed back to the return URL specified in the FORM POST above. The return URL includes details of the purchasing user and the resource being purchased.


PayPal is the safer, ea...
PayPal, Inc. [US] https://www.sandbox.paypal.com/us/cgi-bin/webscr?cmd=_flow&SESSION=SDxahzz0QSu34
Peter Georgeson's Test Store
Resource 1
Total: $0.10 USD
PayPal is the safer, easier way to pay
PayPal
Secure Payments
PayPal securely processes payments for Peter Georgeson's Test Store. You can finish paying in a few clicks.

Step 4—Verify Payment

After payment, the user's browser is redirected to the return URL—the "purchased" page on the sample application, which has the format:

```
http://127.0.0.1:8000/purchased/1/2/?tx=4AF90390YT785663B&st=
    Completed&amt=0.20
```

The URL contains the user ID and the resource ID, which are first extracted in *views.py*:

```
def purchased( request, uid, id ):
    resource = get_object_or_404( models.Resource, pk=id )
    user = get_object_or_404( User, pk=uid )
```

PayPal includes a bunch of extra `GET` parameters on the return URL, which allows us to verify the payment. Without this, it would be easy for a user to manually type in the URL to make it appear as though they had just made a purchase.

The application verifies the payment, first by checking that the transaction ID has not already been used:

```
try:
  existing = models.Purchase.objects.get( tx=tx )
  return render_to_response('error.html',
    { 'error': "Duplicate transaction" },
    context_instance=RequestContext(request) )
except models.Purchase.DoesNotExist:
  ...
```

If the transaction ID is not a duplicate, it is then verified using PayPal's "PDT validation" (described next). If all is well, a Purchase record is added and the user is shown a page indicating a successful purchase, with a link to the original download page:

```
tx = request.REQUEST['tx']
result = paypal.Verify( tx )
if result.success() and resource.price == result.amount(): # valid
  purchase = models.Purchase( resource=resource, purchaser=user,
tx=tx )
  purchase.save()
  return render_to_response('purchased.html',
{ 'resource': resource }, ...
```

An unsuccessful transaction results in an error page, which is easy to generate by changing the transaction ID in the URL.

PDT validation

The sample application uses PDT validation to check that a valid transaction occurred at PayPal.

PDT validation works by PayPal redirecting the user back to your site and your return URL after the transaction has been completed. This return URL has a number of parameters added to it by PayPal. Your site validates the transaction by making an HTTPS request back to PayPal using these extra parameters.

```
class Verify( object ):
  '''builds result, results, response'''
  def __init__( self, tx ):
      ...
      post = dict()
```

```
        post[ 'cmd' ] = '_notify-synch'
        post[ 'tx' ] = tx
        post[ 'at' ] = settings.PAYPAL_PDT_TOKEN
        self.response = urllib.urlopen( settings.PAYPAL_PDT_URL,
urllib.urlencode(post)).read()
        lines = self.response.split( '\n' )
        self.result = lines[0].strip()
        self.results = dict()
        for line in lines[1:]: # skip first line
          linesplit = line.split( '=', 2 )
          if len( linesplit ) == 2:
            self.results[ linesplit[0].strip() ] = urllib.unquote
(linesplit[1].strip())

    def success( self ):
      return self.result == 'SUCCESS' and self.results[ 'payment_status
' ] == 'Completed'
```

An advantage of PDT is that it does not require a public-facing URL, so it will work within your development environment.

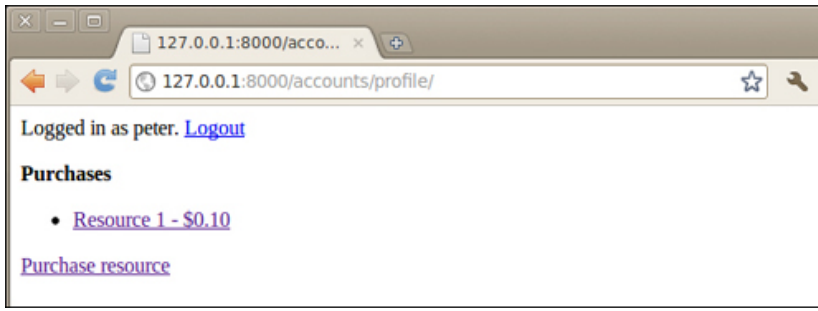
A disadvantage of PDT is that it requires the user to return to your site after payment. If the user makes a payment, but doesn't make it back to your site, the transaction will not be recorded by the application. In such instances, manual intervention would be required for the transaction to be recorded by the application (i.e., you'd need to check your PayPal transactions and manually add the purchase). Hopefully, most users would be keen to return to your site to collect the content they have just purchased.

The alternative to PDT is IPN (Instant Payment Notification). IPN requires a public URL for PayPal to notify your application with transaction details. Unlike PDT, PayPal *guarantees* that your IPN URL will receive notification of a successful transaction. The sample application does not implement IPN, but an IPN implementation is similar to PDT. A recommended approach is to implement both protocols and discard duplicate notifications.

Step 5—Successful Purchase

Once the content has been verified, it is added to the Purchase table. The user can then download this content at any time via their profile page at *http://127.0.0.1:8000/accounts/profile/*:

```
@login_required
def profile( request ):
  '''show resources that a user has purchased'''
  return render_to_response('registration/profile.html',
{ 'list': models.Purchase.objects.filter( purchaser=request.user ) },
 context_instance=RequestContext(request) )
```



Further Enhancements

This simple application of Django and PayPal demonstrates the basic implementation and workflow of an online store offering downloadable content. However, it is not complete and should *not* be used as a live online store without further development:

- A more complete authentication system with sign-up and account management is required.
- IPN should be implemented for robust transaction tracking, to handle the case where a user does not return to your store after purchase.

Conclusion

This tutorial demonstrates a basic workflow for accepting payments with Django using PayPal as the payment provider. It could form the basis for a more complete ecommerce website based on these technologies.

Further Information and Other Resources

- The Django framework: *http://www.djangoproject.com/*
- PayPal Developer Network: *http://www.x.com/*
- Sample application source code: *https://github.com/supernifty/django-paypal-store-example*

CHAPTER 9

Accelerate Your Development Using the Apigee API Console

Bill Day

It seems these days that every service provider must offer a developer API for their services if they are to attract partners. This rise of the API economy has been well documented and discussed for some time. What may be a bit more lacking than APIs themselves, however, are tools to make API usage simpler for the average third-party developer. Apigee is a company bent on changing that.

I briefly introduced Apigee's PayPal API Console in a recent PayPal DevZone blog post. There's much more to be said and shown with the console, however, so let's explore!

What Is the Apigee PayPal API Console?

First up, let's briefly revisit just what the console is and the kinds of things you can do with it.

Apigee announced their free PayPal API console at the Innovate 2010 conference. They noted in their announcement:

> The API Console provides a really easy way to learn, debug, and test the APIs by allowing you to view request and response pairs, drill into errors, and share code snapshots with others.

PayPal Senior Director Damon Hougland provided his own description in a post on The PayPal Blog:

> If you're like me and want to play with APIs before actually writing code, I think you'll really like what our friends at Apigee.com have just built. They've released a new API console that provides a great way for PayPal X developers to send and receive API calls...using various input parameters to see how APIs work and interact

Key point: You use the console to send test requests and it then displays the PayPal development sandbox responses back to you.

The console is in effect a sort of browser-based cURL for Web API programmers. As such, the console lets you very rapidly explore APIs and test out ideas before you code them into your application. (By the way, if you are interested in seeing how cURL itself can be used to make PayPal API calls, refer to Travis Robertson's post "Chained Payments Using NVP and CURL in 4 Easy Steps".)

You can get a good feel for what the console enables you to do and how to use it from this Apigee video demo: *http://www.youtube.com/watch?v=1FKrNQgKSGs*.

APIs Available to You in the Console

Apigee's PayPal console exposes PayPal Adaptive Payments APIs. Calls available within the console include:

- `Pay`
- `PaymentDetails`
- `Preapproval`
- `PreapprovalDetails`
- `CancelPreapproval`
- `ConvertCurrency`
- `Refund`
- `GetPaymentOptions`
- `SetPaymentOptions`
- `ExecutePayment`
- `GetShippingAddresses`
- `GetFundingPlans`

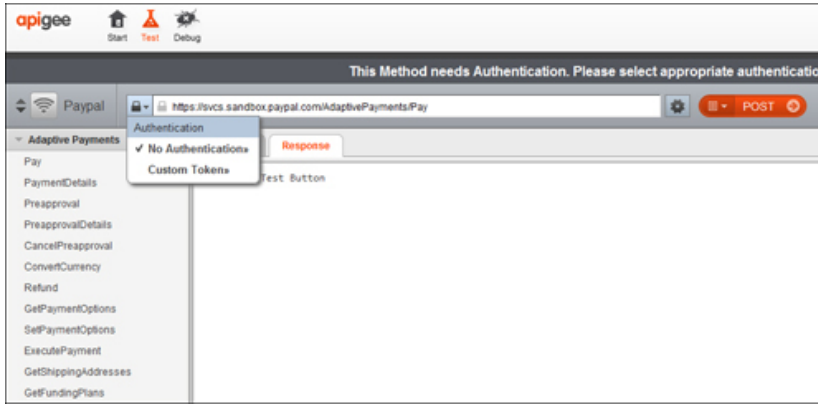
Click here to access the "Adaptive Payments Developer Guide" (PDF format) and learn more about the usage of each of the various API calls available to you.

Using the Console

As was illustrated in the console demo video above, you will need to have a PayPal Sandbox login along with test API credentials in order to make API calls in the console.

If you don't already have these, you should go to *http://bit.ly/gtDs8o*, create a login, then create a test Business Account from the "Test Accounts" page. Creating this business account should automatically create a matching set of test credentials (API username, API password, and Signature) available under the "API Credentials" section selectable on the left side of the screen. These are the credentials that you use in the Apigee console, as shown in the demo video.

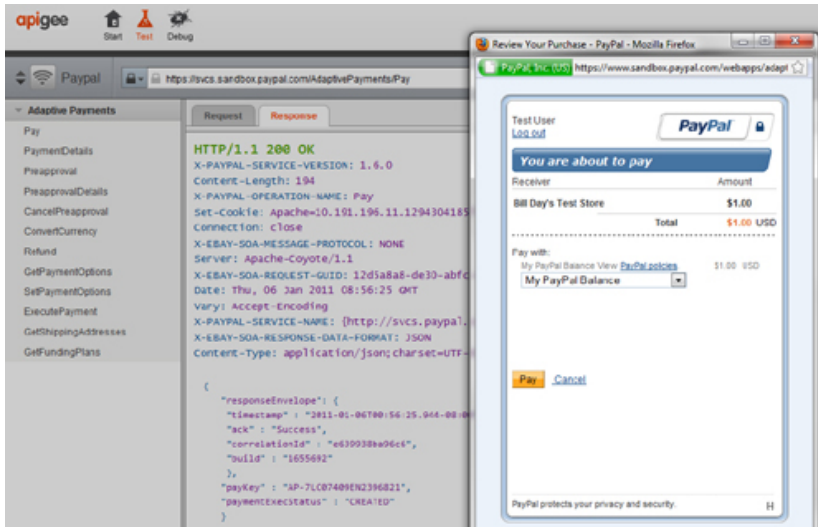
Apigee has updated the console's interface since the video was recorded. In order to enter your credentials into the console, you will need to click on the padlock to the immediate left of the sandbox API call field as pictured below, then select "Custom Token" and enter your credentials as illustrated in the video.



Other changes you may notice in the console interface versus what's pictured in the video:

- "Advanced Settings" (including seller ID in this `Pay` example) are now available by clicking on the sprocket icon to the immediate right of the sandbox API call field.
- Rather than a "Test" button and separate HTTP method selector, the console now has a pulldown list from which you select `GET`, `POST`, `DELETE`, or `PUT`; you then click on the selected method's name to the right of the pulldown to execute it.

Following along with the video, you should receive a response back from the sandbox (see greyed out Response area in the console screenshot below). The Apigee console will also launch a payment window in which you can go through the payment flow. Be sure you login in this payment window using a test buyer account (created on the same developer sandbox page as your test business account).



You should review the information in the original request as well as the response you received back. Both can give you good insight into what exactly is needed to properly use the particular API call of interest. If you've read any other description of how to use the APIs but been confused or missed where a key parameter was set, Apigee's console may help make everything clear by laying all the details bare for your inspection.

Linking Calls Together

Once you have successfully processed one test payment, you might want to perform a follow-on operation in the console.

For instance, suppose you want to get the payment details for the payment you just completed. To do this, you use the console to make a call to `PaymentDetails`.

In order to request the details for the correct payment, you need to set the `payKey` value in the "Advanced Settings" to be the value returned in the `Pay` response. Do that, and you should see details such as the following in the sandbox's second response:

```
HTTP/1.1 200 OK

X-PAYPAL-SERVICE-VERSION:
    1.6.0
Content-Length:
    894
X-PAYPAL-OPERATION-NAME:
    PaymentDetails
Set-Cookie:
    Apache=10.191.196.11.1294305674004749; path=/; expires=Wed,
23-Nov-04 02:52:58 GMT
Connection:
    close
X-EBAY-SOA-MESSAGE-PROTOCOL:
    NONE
Server:
    Apache-Coyote/1.1
X-EBAY-SOA-REQUEST-GUID:
    12d5aa14-3310-abfc-4996-9526fffff5e0
Date:
    Thu, 06 Jan 2011 09:21:14 GMT
Vary:
    Accept-Encoding
X-PAYPAL-SERVICE-NAME:
    {http://svcs.paypal.com/types/ap}AdaptivePayments
X-EBAY-SOA-RESPONSE-DATA-FORMAT:
    JSON
Content-Type:
    application/json;charset=UTF-8

{
   "responseEnvelope": {
    "timestamp" : "2011-01-06T01:21:14.425-08:00",
    "ack" : "Success",
    "correlationId" : "5064810d14d50",
    "build" : "1655692"
    },
```

```
   "cancelUrl" : "http://app.apigee.com/console/-1/handlePaypalCancel
?",
   "currencyCode" : "USD",
   "paymentInfoList": {
    "paymentInfo": [
     {
      "transactionId" : "31X73338M04604710",
      "transactionStatus" : "COMPLETED",
      "receiver": {
       "amount" : "1.00",
       "email" : "p-sell_1294300974_biz@billday.com",
       "primary" : "false"
       },
      "refundedAmount" : "0.00",
      "pendingRefund" : "false",
      "senderTransactionId" : "05K54869K94477535",
      "senderTransactionStatus" : "COMPLETED"
      }
    ]
    },
   "returnUrl" : "http://app.apigee.com/console/-1/handlePaypalReturn",
   "senderEmail" : "sandbx_1289201962_per@billday.com",
   "status" : "COMPLETED",
   "payKey" : "AP-0231274870617083F",
   "actionType" : "PAY",
   "feesPayer" : "EACHRECEIVER",
   "reverseAllParallelPaymentsOnError" : "false",
   "sender": {
    "email" : "sandbx_1289201962_per@billday.com",
    "useCredentials" : "false"
    }
   }
```


There is much more to be learned about the PayPal Adaptive Payments APIs. I'd encourage you to try out each of the calls, with various parameter values, as you explore the console and its capabilities. Then in the future when you are developing an application that uses Adaptive Payments, remember that you can use the console in the same manner to investigate the responses you are receiving back from the sandbox and update your code accorindgly.

Other Apigee Consoles You Might Want to Check Out

One other interesting thing to note is that Apigee provides developer consoles for a number of other Web APIs in addition to PayPal's Adaptive Payments. As of this writing, Apigee also provides free consoles for development against APIs from Facebook, Foursquare, LinkedIn, Salesforce, SimpleGeo, SoundCloud, Twilio, and Twitter.

For example, here's a short video demo of the Apigee Foursquare console: *http://www.youtube.com/watch?v=jIM7EFnSSZQ*.

If you're interested in learning more about or using any of these third-party APIs, you should give Apigee's corresponding consoles a try. Watch for more on these in my blog post series on mobile, social, and local APIs in early 2011.

Accelerate Your Own Development

We've covered a lot in a short amount of time in this article. You were introduced to Apigee and their PayPal API Console and shown how you can use the console to quickly prototype new ideas for your PayPal X Platform development. We also briefly highlighted some of the other APIs for which Apigee provides consoles.

Now it's your turn. Given the low price to try it out (free), why not give at least the Apigee PayPal API Console a spin yourself? And if you do, please leave a comment below letting us know how it went and what you'd like to see added by Apigee in the future to make their console meet your needs even better. Until next time, may your every request receive an ack of "Success"!

About the Author

Bill is Founder of Day Web Development where he provides technical writing and editing, specification development, and platform and technology evangelism services. Recent projects include writing source code level documentation, articles, partner how-to materials, conference session descriptions, and blog entries for O'Reilly Media, NVIDIA, Nokia, and Digital Reasoning Systems. In addition to being an active writer and blogger for the PayPal X Developer Community, Bill is also a contributor to GeekDad.com. Learn more and contact Bill via his LinkedIn profile and BillDay.com.

CHAPTER 10

Interview with iConcessionStand

Travis Robertson

Last Friday, I had the opportunity to speak with Humberto Roa, one of the founders of iConcessionStand. For those who haven't yet heard of the company, iConcessionStand was the winner of the $100k Developer Challenge at the PayPal X Developer Conference in October.

Humberto is one of the nicest people you will ever meet, and he and his team absolutely love working with PayPal. You'll see that come through as he discusses their experience working with the PayPal developer team, the mobile API libraries, and the developer community.

Just one month prior to winning the Developer Challenge, the team at iConcessionStand launched their app at SafeCo Field, the home of the Seattle Mariners. Needless to say, it's been a busy couple of months and they've got a lot more planned as they gear up for a packed 2011.

I hope you enjoy reading this interview as much as I did conducting it.

Travis: Alright, this is Travis Robertson with the PayPal Developer Network and on the line with me is Humberto Roa with iConcessionStand.com. Humberto, thanks so much for joining me.

Humberto: Thanks for having me on, Travis.

Travis: What I want to talk about here today are some of the exciting things going on over at iConcessionStand and let the readers know what's going on and then I know there's going to be a version of this that's going to go around internally at PayPal. So, with that said, why don't we just get started and, for those who aren't familiar with iConcessionStand, why don't you just give us the quick elevator pitch? Tell us what you're doing, what the company does, and what your app does?

Humberto: Sure. So, in general, the product that we're putting together is a marketplace—a location-aware mobile marketplace. And the idea behind the product is that, when you're out, away from your desk, away from your home, at a sports event, or at a concert, or some other location, the vendors at that location can actually make their products available to you through the mobile device so that you can place orders.

So it's a pretty straightforward concept. An example of that is that, in September, we launched the product at SafeCo Field in Seattle, which is home of the Seattle Mariners. And so fans that were attending the games at SafeCo Field were able to download the application onto either their Android or iPhone phones. They were then able to browse the in-seat service menu for food and the team store—or, select items from the team store—and they could browse, create an order, then pay for that order using the mobile libraries. And having the staff from either the team store or the in-seat service bring the food or merchandise back out to your seat. It's a pretty awesome system that got really great feedback from fans.

Travis: That's great. So you guys launched in September. How long have you been in development on it?

Humberto: The company formed up in early 2010. We actually launched the idea at PayPal-sponsored event. I believe it was in April, PayPal sponsored an event called iPad Dev Camp just after the iPads came out. This event was put together to allow developers to spend a weekend at a hack-a-thon, pulling together a concept and on Sunday afternoon each team was given three minutes to pitch the idea to the audience. The idea we put together was a very early, rough concept around this and we gave the pitch. We ended up winning best use of PayPal's APIs at the conference so we each got iPads out of it.

Travis: Nice. Now is this an idea that you had been having for a while and just felt like, "If only the technology was there..." or is it something once the technology came out, you realized this would be a great use of it?

Humberto: It's kind of a funny story where we came up with the idea. We signed up for the Dev Camp but we didn't know what we were going to do. We were kind of having a brain storm the night before and the wife was there and said "You know, it'd be really fun if I didn't have to get up from my seat to place orders. I wouldn't have to haul kids with me." And we kind of thought "Hey that sounds pretty fun and we laid out the whole design that Saturday morning at Dev Camp. We worked on it, it's been about, well we slept four hours that night, on Saturday night and between Saturday morning and the pitch, that was it. We were working continuously. So there was three of us working on the concept and we were pretty excited with the feedback that PayPal gave us. That was kind of the springboard that launched the idea.

Travis: From there did you just kind of keep the ball rolling and decide "Let's just make this a full on company?"

Humberto: Yeah, that's exactly it. We grew the bean. We got some guys on that had done similar, not similar but had experience in the past with eCommerce systems or scaleable server infrastructures. And as a team, we started from scratch so we made a completely new product and we pitched the product to the Mariners. It lined up with some initiatives that they had and they gave us the opportunity in September to try it out. So, in September with a completely new product for Android and iPhone, we actually ran live at 12 Mariner's home games.

The way it worked at Mariner's home games, it was kind of an interesting story because as part of the pilot, the conditions of the pilot, is that we were not allowed to do any marketing behind it. We weren't allowed to do any Facebook or Twitter or do any kind of press releases. So the way that the fan in the audience found out about the application is on the big screen at the top of the second inning, an advertisement would drop to let the fans know they could order from their seat. Basically we put a URL up on the big screen that they could point their mobile device to and depending on the device type, we would forward them to the right app store to get the app and download it to their phone. Literally, that one advertisement is the only way that fans found out about us.

Travis: How was the response?

Humberto: The response was great. We got a lot of feedback. In fact, the Mariners were surprised because some of their internal staff actually used the system to place orders and they were getting really good internal feedback. So they were really excited about that and we feel that overall it went really well.

Travis: So did they [the Mariners] have to staff up for your app going out just because they were opening it up to a larger percentage of the stadium? Or is it something that they thought, you know, let's just see what the response is and they ran it with their regular staff levels?

Humberto: They ran it with the regular staff levels so there was no staffing adjustments made based on the application.

Travis: OK. Interesting. So let's talk a little bit about how you guys are using the Mobile Payments Library. I know that the application does rely on it, but can you kind of tell me about your experience getting it integrated into the app? You know, specifically how you're using it, just kind of what your experience has been with it?

Humberto: Sure. So our first experience with the Mobile Payments Library was at the iPad Dev Camp. The iPad Dev Camp was hosted at PayPal's headquarters, and as part of the event PayPal made available a very early version of the Mobile Payment Libraries for developers to use in their projects at the camp.

With our idea, we actually integrated in against the Mobile Payment Libraries directly at the camp, and that first experience was actually less than an hour to get that library integrated and have it working and be able to run the payments through the system as part of the ordering process. It was really fast.

And then, when we took the products to the next step to be used at SafeCo Field, we didn't find the process much different. We covered some more conditions and edge cases around our own abilities. So handling cases where the payment status was pending or if the payment needed to be canceled and refunded. So we broadened our use cases a little bit. It was more incremental work using the PayPal libraries. It wasn't a complete ground-up rewrite for payments. It was a great experience overall integrating with the libraries.

And the thing that's exciting for us with the libraries is PayPal's handling all the complexities that normally you'd have to deal with launching a system like this. So, aspects around the PTI compliance or the customer's privacy and in other ways that PayPal either maintains or manages that data put the Mariners, for example, at ease with working with a company of our size.

It was an awesome experience overall.

Travis: There are really two parts to the app. There's what the consumer uses to place an order and there's the order fulfillment side of it. Can you talk to us about that order fulfillment side and if and how PayPal integrates into that part of it as well?

Humberto: Well, in general we have a very standard order workflow where we get the cart received from the user, we verify the cart in a number of ways. The values, the inventory quantities are still available and then we make the request to PayPal to process payment and when the payment's successful, then we move the order to another step where the fulfillment staff becomes aware of the order. They are able to prepare the order and when the order has been delivered, they're able to mark the order as delivered in our system. So it is a straightforward workflow.

The interesting things that would differentiate it from a standard ecommerce system is that it's "at location" fulfillment. If you think about an Amazon.com, they're going to receive an order and they're going to have a few days to fill the order and get it shipped and there's a number of steps involved. With the system that we were doing, the order was placed and within thirty minutes of

the order being placed, the product was delivered to that person at their location within the stadium. You basically have the order, the payment, the preparation of the food and the delivery occur within a very compressed time scale.

Travis: And so on the order fulfillment side of it, is there an iConcessionStand application that they're running or how do they get notified? Do they get notified in existing systems or is it something where they're using an existing system and using an iConcessionStand fulfillment system?

Humberto: We take the order from end to end. From the time that the customer downloads the application to the time that the customer receives the food, all the steps that take place that are managed are completely within the iConcessionStand system.

But we can also sit along side other systems if the customer already has an existing order processing system or some other aspect of that flow, we can tie directly into it. And that's been a big difference with the people that we've pulled together for the team. So we all have experience in enterprise software integration at very large companies. That aspect was an aspect that we designed in early, and it's incorporated from even the very beginnings of the product that we built after iPad Dev Camp. To be able to swap components out with solutions that the customer may already have or that the vendors that we're working with already have. I think that makes us unique in our offering because for small vendors who don't have any existing infrastructure, we can take the process end to end completely but for large companies who've already made some investments, we can leverage those existing investments but bring in a whole new sales channel through the mobile device.

Travis: I want to talk to you about your experience at the PayPal X Developer Conference and, specifically, you guys won $100,000. Talk to me about that experience. Has it sunk in yet? Is it surreal? Share with us your experience at the conference and with the challenge.

Humberto: It was overwhelming and we're still kind of in a state of shock over what went on at the end of October and we're just so excited about the vote of confidence that we got in the concept as part of winning the 2010 developer challenge. So that was a huge boost in a lot of ways. It gave us additional capital that we could use to kind of go into other markets.

Travis: So you didn't buy a Mercedes then?

Humberto: No. We actually re-invested everything back in.

Travis: Oh, what fun is that?

Humberto: It was just awesome and not only for the working capital. But we're now getting the word out about what we're able to do and how PayPal in general is supporting us in this venture and into a new area. It has been great working with potential customers. So as we're engaging with new sports teams or new vendors in other markets, we come in with an immediate advantage and the ability to put them at ease in a lot of ways.

It's just been an awesome experience for us.

Travis: Now, had you guys been to the 2009 developer conference or was this your first year?

Humberto: This was our first year. It was our first experience at a PayPal conference. The conference in general, we were really excited even to just go to the conference. I think the break out sessions and the speakers that PayPal had available to developers, it was just an awesome experience.

Travis: Did you guys, as you were going through some of those different break out sessions, did you come away with additional ideas, additional insights, things that you thought "We really gotta start thinking about how to integrate that particular technology or that particular idea into iConcessionStand moving forward?"

Humberto: Absolutely. And even having access to the PayPal technical team helped even more. So not only were we able to brainstorm on ideas of new technology coming out from PayPal, we were able to go right to the source of that feature and ask him and brainstorm with him how we could use it in our own application.

PayPal went out of their way to make people available. People were walking around with T-shirts, every person from PayPal on the back of their T-shirt said ask me. You have a question, ask me. I'm available. Or something like that. I think they way that they tried to open up and bring the developers into a tight relationship was just amazing. I haven't experienced that with another company.

Travis: Has that continued for you guys as you've continued working with PayPal, moving beyond just the conference? How's the developer community been? How's the PayPal community been in terms of just general support with what you guys are building?

Humberto: The support's been great. We are actually doing things in our product that are a very non-standard use case just because the market's not very mature in some ways with these type of products, so we've been able to get the support to think through the problem and to be able to pick the best approach at solving it. PayPal support's been very awesome. The developer community has been great in helping us vet ideas and make sure that the

products that move forward provide the best experience for the vendors and the vendor's customers.

Travis: Well you know what, Humberto, I've taken a lot of your time here and I really appreciate it but before I let you go, I'd love to find out what you can say on the record about what can people be watching for over the next year or maybe you want to go out a few months but just what can we be looking for from iConcessionStand? Is this something I'm going to see at Tennessee Titans stadium anytime soon or maybe at the Nashville Predators hopefully?

Humberto: Well we can't talk about the specific customers we're working with but I can say we're pretty excited about new customers we're going to be launching with this upcoming year. So all of the guys on the team are huge sports fans, so this is really turning into a dream job of providing a product that we're experts at, that we know the customers enjoy and bringing that into an industry that we love.

It's a great combination. I can say that it's going to be pretty exciting with the sports teams that we're going to be launching with in the upcoming year and also the new markets that we're taking this product out to. We're going to make that information available when it's appropriate.

Travis: As it makes sense, sure.

Humberto: Yeah, we're looking forward to announcing some really great news over even the next six months.

Travis: Awesome. Any parting thoughts? Anything you want to leave the audience here before I let you get going?

Humberto: Yeah, I would come back to the parallel payments. I think that PayPal's made some really big advancements in the ability for you to monetize your own products so with the parallel payments, you can facilitate transactions with up to six people. You can hide five out of that six people or you can expose all six people to the customer at the time the payment's received. I would encourage developers out there to kind of rethink the way that they're monetizing their sites. See if there's an opportunity to approach the site from a marketplace perspective. If you can improve the experience between the buyer and the seller, there's an opportunity in there for you to take a cut of that transaction and build a revenue model around that.

Travis: And that's pretty much what you guys have done.

Humberto: Yeah, and I will say this, that without the PayPal mobile products, these types of transactions occurring on a mobile device are very, very complicated accounting-wise and PayPal's really simplified that model. I mean I wouldn't be surprised that within the next year, marketplaces and various industries start popping up for developers using these new libraries. PayPal

has done a really great job at thinking through what's required from a developer's perspective to launch these type of marketplaces.

Travis: Just one final question. What are your big, bold predictions here for mobile payments as you see them for 2011?

Humberto: I would think it's not a stretch to think that you could go into a storefront, find a product, pull out your phone, make the payment and be able to walk out of the store with that product, with the transaction being completed. So I think that where these payments are going on mobile devices is in the next two to three years, at some point it's going to be rare for you to actually have to stand in line to complete your payment at a physical store front.

I think it's going to go in a direction where you can handle a whole transaction, everything that's involved, directly from your phone. I think a good model of where that is now is, for example if you go to an Apple store they come to you and they bring the device to you and you complete the transaction on a one-to-one basis. And I think that, you know, within the next two to three years, as the technology is approved on the mobile devices, you know, it might just be you and your device. That's all you need to complete the transaction. Yeah and maybe you just show a confirmation of the transaction as you walk out. Personally I think that's the direction, or one of the directions that this, that the mobile devices are going. So, I'm pretty excited about it personally because, I mean, if you've been at a store and, you know, you want to get out of that store, but the line is like 50 people deep, you know, it's just miserable waiting in that line.

Travis: I can see it being huge during the holiday shopping every year, when those lines just get miserable at the mall.

Humberto: Yeah, and I think it's going to challenge retailers to rethink their models. I mean, maybe you don't want it across everything, but I bet there's product lines that are going to be very good product lines for this type of payment and, you know, particular types of merchants who are going to be able to provide a really great new type of experience for their own customers.

Travis: Yeah, yep, well Humberto thank you so much for joining us and giving us your time. I know you're busy preparing for some pretty exciting stuff coming up, so we really do appreciate you taking the time and look forward to kind of following your progress and that of iConcessionStand.

Humberto: Hey, thanks a lot Travis, it was really great talking to you today.

About the Author

Travis Robertson is a business strategist and consultant. He also writes about entrepreneurship and leadership on his blog TravisRobertson.com. Read more from him on his X.com blog or follow him on Twitter (@travisro) where he spends far too much time.

CHAPTER 11

Magento, Part 1: How to Install Magento

Travis Robertson

Here at PayPal and the X.com developer community, we're very excited about PayPal's recent purchase of the Magento ecommerce platform. There is so much change happening here on X.com, and much more is planned for the future.

To celebrate, I'm launching into a rather extensive and long series on Magento. The system is extremely powerful, and there is a lot to cover. But, to start exploring, you need to have an installation of Magento that you can test with or experiment with, so that's what we'll set up here today. Then, as we progress through this series, you'll be able to tweak and modify your installation as necessary—testing out themes, extensions, tools, configuration options, and more.

Comparing the Editions

For this article, we will use the free edition of Magento, called the Community Edition. If you're just starting out with Magento or if your needs aren't too substantial, this is really a great way to get started. Keep in mind that you're on your own when you roll with the Community Edition; you don't get the luxury of support that you would with either the Professional Edition ($2,995/year) or the Enterprise Edition ($12,900/year).

There are very few differences between the Community Edition and the Professional Edition in terms of functionality. You're really paying for enhanced security, support, and a few other features that may or may not be important for you depending on your specific needs. The major difference in functionality

comes with the Enterprise Edition, which is really intended for major ecommerce websites.

If you're interested in seeing the full comparison between editions, check out this page on the Magento site for a nice little chart.

Server Requirements

Before walking through all of this setup, you want to make sure your server will support Magento. The basic requirements are as follows:

- The server must be running Linux. No Windows servers allowed.
- Apache 1.3.x or Apache 2 for the web server.
- MySQL 4.1.20 or above.
- PHP 5.2.0 or above, and safe mode must be turned OFF.

If you have other concerns about whether your server is compatible, head over to this page, which will give you a script you can use to check compatibility for your server setup.

Installing Magento, Step 1: Download, Extract, and Upload

There are a couple of ways you can install Magento; the first is to upload the extracted files via FTP, and the second is to use SSH to download and extract the files to your server. Because there are still a number of hosting companies that do not provide SSH access to servers, we're going to handle installation the old-fashioned way: option one.

Go to *http://www.magentocommerce.com/download* and select the appropriate download options based on what you're looking for. You'll want to download the file under the section labeled Full Release and then extract the files prior to uploading them to the server. Once you've done this, open up your favorite FTP client, connect to your web server, and upload the extracted folder (called *magento*) on to the server in your website's *root* folder.

When it's uploaded, the URL structure will be something like *http://www.yourfancywebsite.com/magento*.

Magento offers some sample data that you can load into the installation if you so desire. If this is your first time installing Magento, I recommend that you take advantage of this option, as it will help you get up to speed more quickly with what's going on in the system. You can find the download for the sample

data on the same page where you downloaded the Magento installation, just a bit farther down.

Installing Magento, Step 2: Set Up Permissions

This next step is very important. The Magento setup wizard needs to have permission to create files and folders where necessary, so you'll want to check your server's permission levels before proceeding to Step 4 (you can do Step 3 without doing Step 2, but not Step 4).

You must set the following files and folders to be writeable (i.e., permission should be set to 777):

- The file *magento/var/.htaccess*
- The directory *magento/app/etc*
- The directory *magento/var*
- All directories under *magento/media*

Installing Magento, Step 3: Creating the Database

Most hosting providers have some sort of MySQL admin console, such as phpMyAdmin or something similar, to allow you to create a database on the server. You'll want to check your hosting company's help files if you're unsure of how to set this up.

Make sure the database user that you either create or use has full permission to the database you set up; then, jot down the database name, username, and password, as we'll need them in the next step.

In Step 1, I mentioned that Magento has sample data that can be loaded into the system and told you where to grab it. Now is the time to add that into the database. You must do this before running the setup wizard in Step 4.

When you extract the sample data, you'll get a *.sql* file and a folder called *media* that you'll be working with. First, import the data in the *.sql* file into the empty database that you created earlier, using the method of your choice.

Next, you'll want to fire up your FTP client and upload the *media* folder into the *magento* folder on your server. So, the URL structure would look like this: *http://www.`yourfancysite`.com/magento/media*. Set the permissions on the *media* folder (and all subfolders) to be writeable (777).

Prior to importing your data, you will need to make sure that your database is completely empty and that no tables exist, or the import will fail. That's why you must do the import before running the setup wizard. If you get an error when trying to import the data, this is probably why, and you will need to go in and drop all of your existing tables.

Installing Magento, Step 4: Run the Setup Wizard

If you've jumped to this step immediately after uploading the files via FTP, take a quick breath, relax, and go back to Steps 2 and 3 first. Not doing so could create headaches and problems for you.

If you're ready to move forward, open a browser and head to *http://www.yourfancysite.com/magento*, and the installation wizard will start right up (very similar to WordPress, if you're familiar with that platform).

Most of the settings in these next steps can be changed later in the Admin console, so don't worry too much about making a mistake. However, I'll walk you through some of the trickier items.

The first thing you'll need to do is accept the terms and conditions before you can continue. Once that's done, you'll need to configure the localization settings, which (hopefully) you know how to do.

Next up is the database configuration. For most hosting companies, the Host value will be `localhost`; however, if you're unsure, check with your host. The database name, username, and password are values you set up earlier in Step 3, so go ahead and enter them here.

Following the database configurations are the Web Access options. The Base URL will likely be prepopulated for you and shouldn't need to be changed. For those of you installing locally, you may have to use the 127.0.0.1 IP address instead of localhost, depending on your setup.

If you don't like the idea of having the path to the admin console being located at *yourfancysite.com/magento/admin*, change it to your liking here. However, unless you have a really compelling reason for why you think it should be changed, I recommend leaving it set to the default.

Next, check Web Server (Apache) Rewrites and Secure URLs. When you see the Session Storage Options, you'll most likely want to use "file system" instead of "database." The only time you'll need to consider the "database" option is when your site will be set up to run on multiple web servers (i.e., high-traffic websites). In this case, you'll need the session data to be stored at the database level so that your site can serve up pages and data from any server without the user losing session information. Filesystem session storage is much faster and will also keep your database from swelling in size.

Finally, you'll need to set up the Admin user (you can add more later). In this section, you will be asked to either create an Encryption Key or let Magento generate one for you. It's your choice here, but I always recommend letting the system generate it.

Proceed to the next step, and you're good to go!

If you ever need to (or want to) walk through the setup wizard again, simply locate the file *magento/app/etc/local.xml* and delete it. You can then return to *yourfancysite.com/magento*, and the setup wizard will appear, allowing you to walk through the installation steps.

Installing Magento, Step 5: Verify the Installation

To see your new site from the visitor's perspective, go to *http://www.yourfancysite.com/magento* and start exploring. If you set up the sample data, you should see some of that on the page.

Then, get familiar with the admin section by navigating to *http://www.yourfancysite.com/magento/admin* and logging in using the data you set up during installation.

Next Steps

We're going to spend a lot of time looking at Magento in the coming weeks, and this installation is a great place for you to test the various themes, extensions, and configuration options we will discuss.

If you're interested in having me cover a particular angle on Magento, let me know in the comments on the article's page at x.com, and I'll try to get it added to the lineup. Also, if you're setting up a Magento installation (or already have one up) and want to share it with the X.com developer community, put it in the comments section as well.

About the Author

Travis Robertson is a peak performance coach and business strategist. He writes about entrepreneurship, leadership, and millennials on his blog, TravisRobertson.com. You can follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 12

Magento, Part 2: How to Set Up Your Magento Store

Travis Robertson

If you're just joining us, this is the second part of the multipart series I'm doing about the Magento platform. PayPal recently acquired the ecommerce platform and there is so much this powerful platform is capable of doing for your business—especially when used in conjunction with PayPal.

In the first part of the series, we looked at how to install Magento, in Chapter 11. If you missed it or haven't yet had a chance to get it installed, click here to go back and walk through the tutorial. You're going to need a working installation of Magento for the rest of this series (and because a lot of articles and blog posts on the new X.com will center around Magento).

In this chapter, we're going to take a look at getting products into your new store and setting up payments with PayPal. So, without further ado, let's dive in.

Creating Categories

Before adding a product, it is important to create and establish categories for them. These categories could be differentiated by the kind of product, a brand name, or anything that distinguishes one product from another.

From the dashboard, hover the mouse cursor over the "Catalog" tab and click on "Manage Categories" from the drop-down menu. This will open the section of Magento where we can find the different categories within our store—or, as is the case here, create a new category, since this is a newly installed Magento store.

The left sidebar of the window displays options to "Add Root Category" and "Add Subcategory". The options "Collapse All" and "Expand All" are viewing options that will come in handy once you have a number of product categories on your store.

There is a default category folder already created. It is a "root" category that you can edit as the first or main category. Click on the folder to edit the settings.

When filling in the fields, try to make your wording as succinct, unique, and descriptive as possible. Here are some of the fields available to you. This list is found on the "General Information" tab:

- **Name**—The name of the category. (This is a required filed)
- **Is Active**—Drop-down options: Yes / No (This is a required filed)
- **Thumbnail Image**—You will be prompted to choose an image from your computer by clicking "Browse." Remember, this is an image for the category, not a given product. You are free to use a product image here but you may want to use something more generic.
- **Description**—Place a description of the category being created.
- **Image**—This image will be placed along with the description when viewing the categories from the front end. Remember that this is a different image from the thumbnail image. However, you may use the same image should you choose to.
- **Page Title**—This will be the title that appears from the front end or store view.
- **Meta Keywords**—Place keywords that describes the category you are creating.
- **Meta Description**—A short description of the category you are creating.
- **Include in Navigation Menu: (Options Yes/No)**—This option will not work if you are creating a "root" category. However, if you are creating a subcategory, choosing "Yes" will make the category visible as a link from the main menu from the front end.

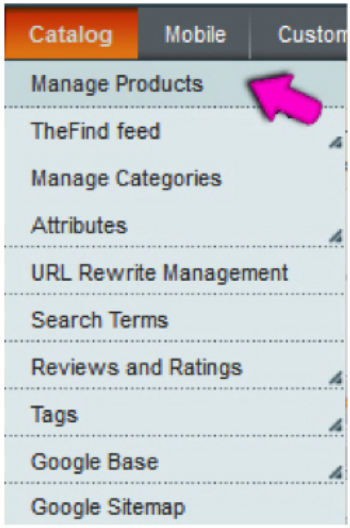
If you want to create a subcategory, the process is effectively the same, with one minor difference. You'll need to make sure to select the parent category you are creating a subcategory for, before clicking on "Add Subcategory." After this, the process of creating a subcategory will be the same as it is for the category.

If you accidentally create a subcategory (or subcategories) under a different category parent than you intended, they can be easily changed by dragging the folders to their intended parent folder.

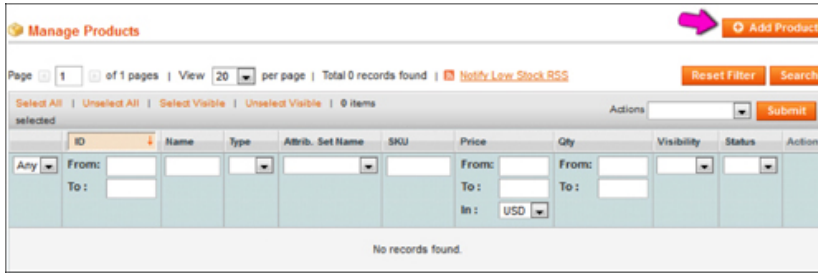
Adding a Product in your Magento Store

Now that you have a category (or categories) created, it's time to add in your products. If you jumped here and have not yet created categories, you'll need to go back and do that first. The Magento system requires that every product be assigned to a category when it is created. Even if you will have only one product in a given category, it is still mandatory to create a category for that product.

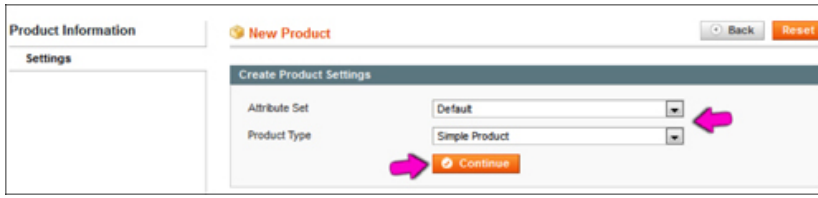
From the dashboard of your Magento control panel, hover your mouse cursor over the "Catalog" link and select "Manage Products" from the drop-down menu. You will now be directed to a menu where you can add, edit, or place several properties or information about your product(s).



From this window, click on the "Add Product" button located on the top-right side of your "Manage Products" window. The image shown below displays no current product(s) on the inventory. However, this is the same location where your existing products are displayed and can be updated.



After clicking the "Add Product" button, you will be prompted to choose two of your product's settings: "Attribute Set" and "Product Type." There are several options under "Product Type." However, for this illustration, we will be choosing "Simple Product." Then click on "Continue."

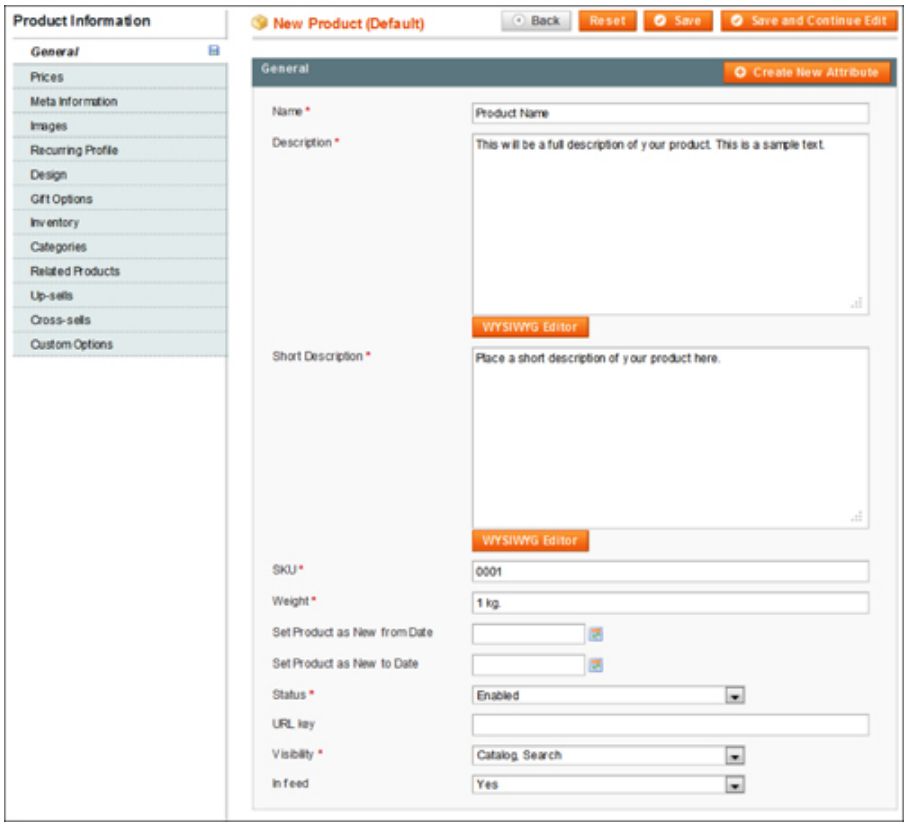


The window that appears next is the new product information window. Under the "General" tab, fill in the fields with their corresponding information. The red asterisk on the field names means it is required and cannot be left blank or without any value:

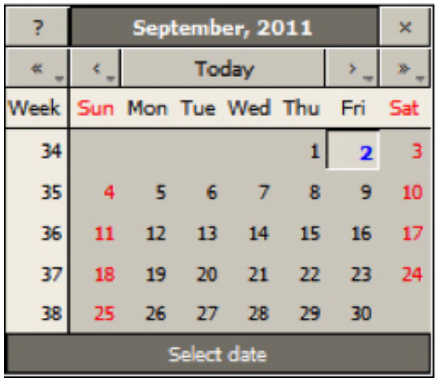
SKU—This is a value that you can place that will point to the product you are placing. The value "0001" is an example—you may also use an alphanumeric value here if need be.

Visibility—This determines where this product can be found. "Catalog" means the product can be found from the front end or your store front. "Search" means your product can be listed from the search field from the front end. If not selected, the search queries will not list this item of the search results. What is selected above is "Catalog, Search", which means both settings are on.

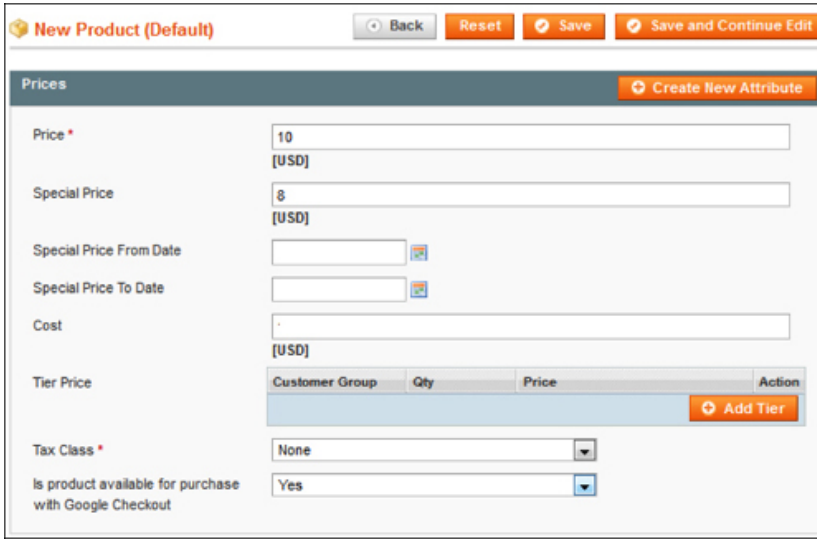
In Feed—The options available are Yes/No, and specify whether you would like to include this product in your website store's RSS feed.



Set Product as New from Date/to Date—This is an option to display your product as "New." You can use this to feature your latest product on the site. "Set Product as New from Date" is the starting date and "Set Product as New to Date" ends the product featured as new. Leaving the date blank will let your product simply be added to your inventory. This is a great feature if you have a hot new product (such as the iPad 3) that you want to prominently display on your site.



After filling in the "General" information, click on "Prices" next. It will display another form like the one below.



Fill in the necessary fields for the price of your product. You'll notice that you can run sales specials on a product by entering in the sale price and the dates for which that sale is active. If you want to track the margins on your products and sales, enter in your cost in the form for each product on this page.

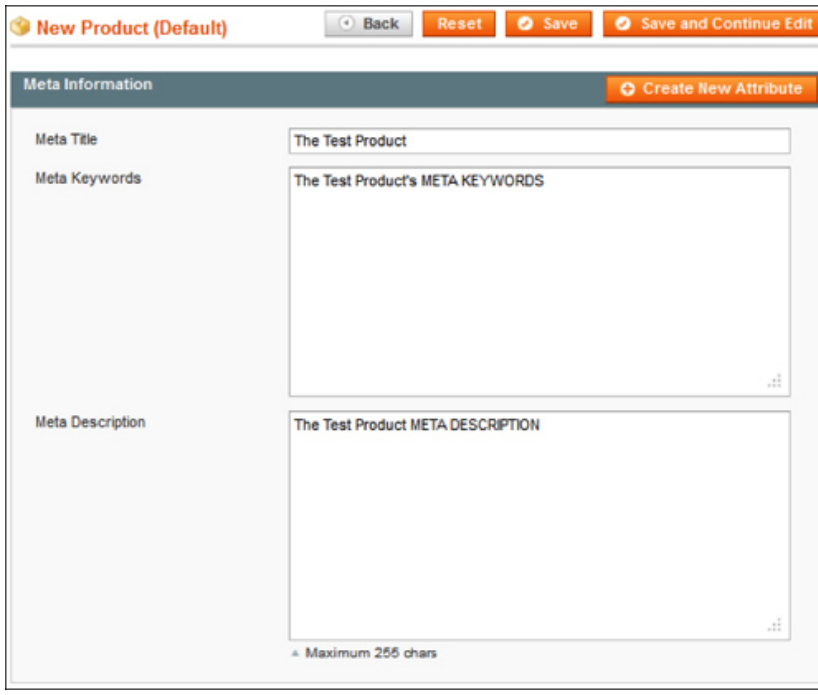
We'll cover reporting in a future post.

Finally, if you have tiered pricing based on either a customer group or on the quantity purchased, you can use the "Tiered Pricing" field to fill in that information.

You'll notice that Google Checkout is the default payment option on this page. In the next chapter, we'll look at integrating PayPal into the payment process, as well as some additional payment options.

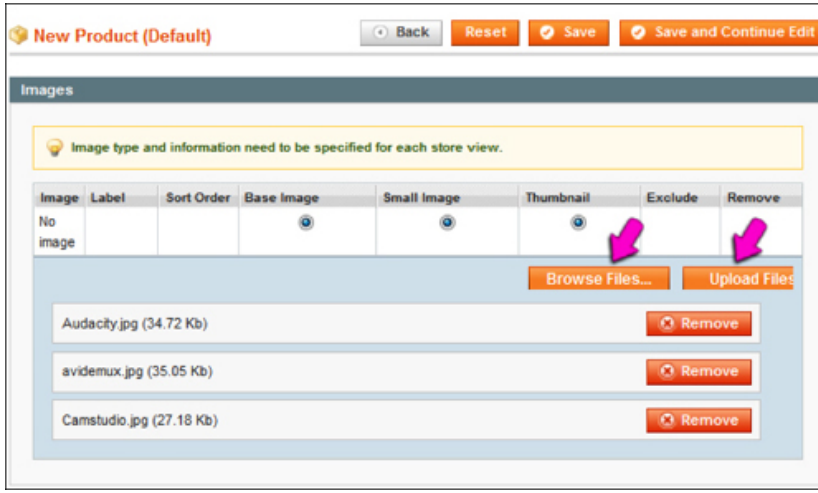
The next tab is "Meta Information." It's important to fill this out because this information can make it known on the Internet that you have this product in your website store.

For the "Meta Title," you may simply enter the Product's name, just like in the "General" information page. Fill in "Meta Keywords" with several descriptive words of your product. For the "Meta Description," enter a brief description of the product. It has a maximum of 255 characters.



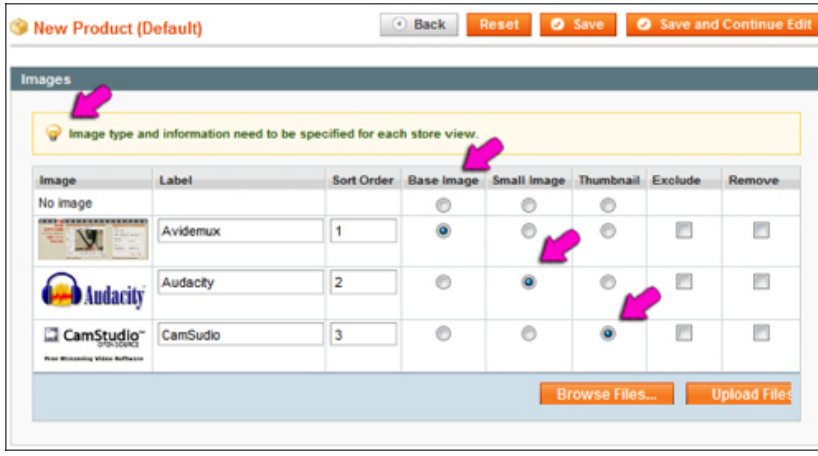
After filling in the information fields, click on "Images."

The "Images" tab displays the form where you may upload images of your product. The image below indicates there are already image files ready to be uploaded. Click on the button "Browse Files" to open a window to search and select the image(s) of your product. In this example, three images are selected and ready to be uploaded. If you have uploaded the incorrect image, simply click the "Remove" button to remove the image from the upload list.

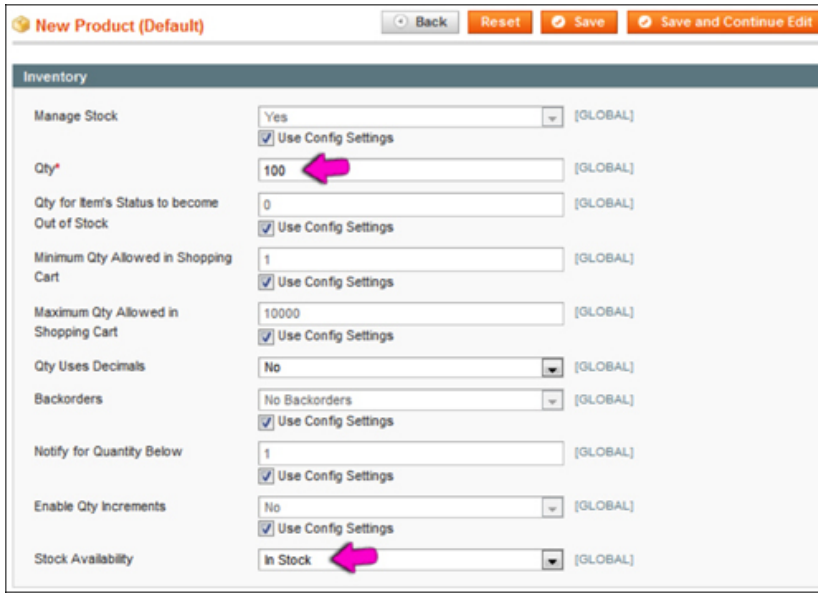


After clicking the "Upload Files" button, the files will now be uploaded to your store. If the upload is successful, it should be able to display them as shown in the next image.

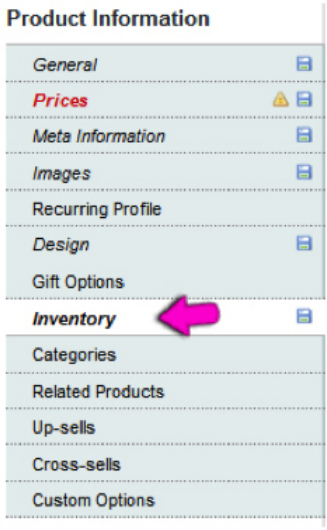
The three sample images that were uploaded can be used as a "Base Image," "Small Image," and "Thumbnail." All three may also be used for one of these options. The label may differ and it is best to specify where each image is used —or, in this case, what it is about. It will help you keep the images straight when you come back to the product.



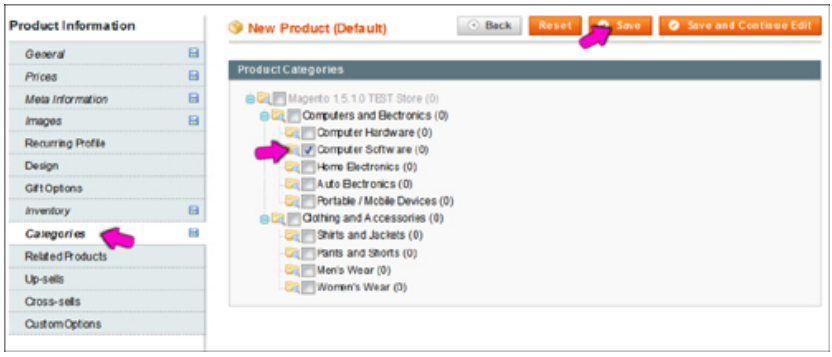
The next tab we will cover is the "Inventory" tab. These settings include the "Stock Availability" and the quantity (Qty) of the product which, is a required field.



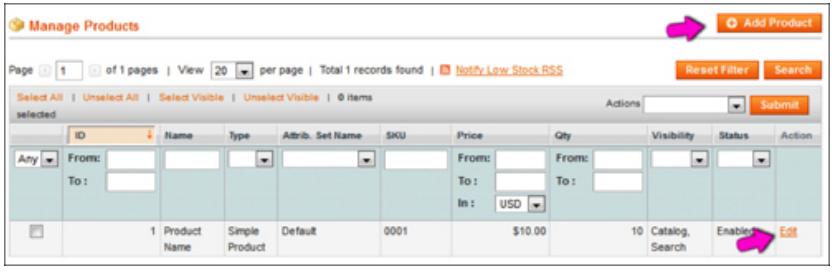
The checkbox below each field indicates whether you will be using the feature, as shown below.



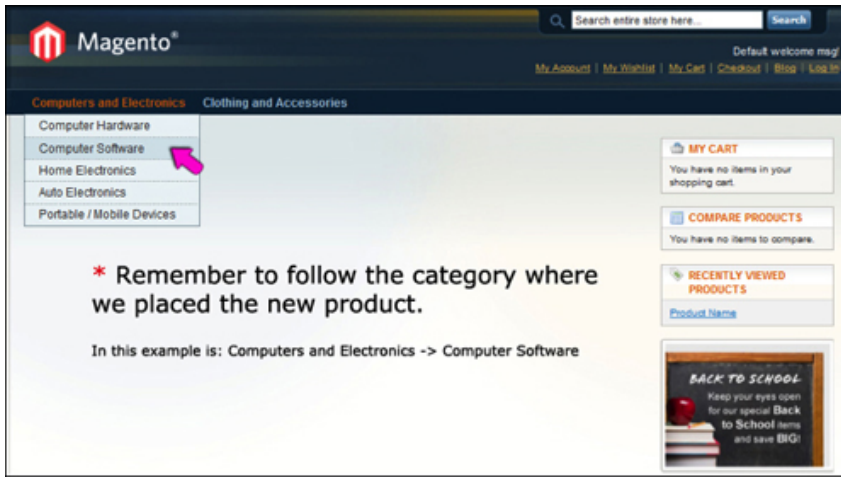
The next step is the “Categories” tab, where we will choose what category our product falls under. The sample images we used are logos from some software. As a sample, we will click on the “Computer Software” category.



As you notice, we choose categories using checkboxes. That means we can select multiple categories for our product under all the categories where it applies. After choosing which category in which to place the product, you may now click on the “Save” button. This will lead you to the “Manage Product” window that now displays the item or product in the list.



From this point, we can continue editing the product we entered or add a new one. We can also check to see if the product we entered displays from the front end by going to the main page of our store. If we see the product there, then we successfully added the product into the system. If not, we need to go back to the “Manage Products” page to see if we might have missed a field or a setting.


Magento
Search entire store here...
Search
Default welcome msg!
My Account | My Wishlist | My Cart | Checkout | Blog | Log In
Computers and Electronics
Clothing and Accessories
Computer Hardware
Computer Software
Home Electronics
Auto Electronics
Portable / Mobile Devices
* Remember to follow the category where we placed the new product.
In this example is: Computers and Electronics -> Computer Software
MY CART
You have no items in your shopping cart.
COMPARE PRODUCTS
You have no items to compare.
RECENTLY VIEWED PRODUCTS
Product Name
BACK TO SCHOOL
Keep your eyes open for our special Back to School items and save BIG!

The first image below shows our store view after clicking the "Computer Software" category, where our product is placed. The second image is the details view, after we click on the product from its main category to get more information.

Wrapping Up

In our next article, we're going to look at how to take payment, and integrate PayPal and a few other payment options into Magento. In the meantime, get your products entered into the system and ready to go.

About the Author

Travis Robertson is a peak performance coach and business strategist. He writes about entrepreneurship, leadership, and millennials on his blog TravisRobertson.com. You can follow him on Twitter @travisro, where he spends far too much time.

Magento
Search entire store here...
Search
Computers and Electronics
Clothing and Accessories
Home / Computers and Electronics / Computer Software
Did you know?
Our customer service is available 24/7
(555) 555-0123
Computer Software
MY CART
You have no items in your shopping cart.
COMPARE PRODUCTS
You have no items to compare.
RECENTLY VIEWED PRODUCTS
Product Name
NEWSLETTER
Sign Up for Our Newsletter:
Subscribe
This is another subcategory under Computer and Electronics -> Computer Software. This is a sample Text. This is another subcategory under Computer and Electronics -> Computer Software. This is a sample Text. This is another subcategory under Computer and Electronics -> Computer Software. This is a sample Text.
1 Item(s)
Show 9 per page
Views: Grid List
Sort By: Position
Product Name
$10.00
Add to Cart
Add to Wishlist
Add to Compare
BACK TO SCHOOL
COMMUNITY POLL
What is your favorite color
Green
Red
Black
Magenta
Vote
NOW ACCEPTING PayPal
About Us | Customer Service
Site Map | Search Terms | Advanced Search | Contact Us
Help Us to Keep Magento Healthy - Report All Bugs (ver. 1.5.1.0)
© 2008 Magento Demo Store. All Rights Reserved.

Magento
Search entire store here...
Search
Default welcome msg!
My Account | My Wishlist | My Cart | Checkout | Blog | Log In
Computers and Electronics
Clothing and Accessories
Home / Computers and Electronics / Computer Software / Product Name
Audacity
MORE VIEWS
Product Name
Email to a Friend
Be the first to review this product
Availability: In stock
$10.00
Qty: 1
Add to Cart
OR
Add to Wishlist
Add to Compare
Quick Overview
Product Name Short Description
Details
Product Name Full Description
Product Tags
Add Your Tags:
Add Tags
Use spaces to separate tags. Use single quotes (') for phrases.
MY CART
You have no items in your shopping cart.
COMPARE PRODUCTS
You have no items to compare.
BACK TO SCHOOL
Keep your eyes open for our special Back to School items and save BIG!
COMMUNITY POLL
What is your favorite color
Green
Red
Black
Magenta
Vote
NOW ACCEPTING
PayPal
About Us | Customer Service
Site Map | Search Terms | Advanced Search | Contact Us
Help Us to Keep Magento Healthy - Report All Bugs (ver. 1.5.1.0)
© 2008 Magento Demo Store. All Rights Reserved.

CHAPTER 13

Magento, Part 3: Accepting PayPal Payments

Travis Robertson

Magento is an extremely powerful, open source ecommerce platform that was recently acquired by eBay, the parent company of PayPal. In my current series of articles, I'm covering Magento's various components and how you can make use of the platform for your own ecommerce needs.

In Part 1 of this series, I took you step by step through the installation and setup of Magento. I pointed out some of the gotchas along the way and the various configuration options available to you. When we were done, we had a working (albeit a rather plain) ecommerce site set up.

In Part 2, we looked at getting products into your new Magento store. We discussed how Magento organizes products and inventory into categories and subcategories, and covered some of the potential configuration options. In that article, I also gave you some homework: getting your products into your new system so that we can begin the process of accepting payment.

And that's where we find ourselves today. After all, what good is a store if you have a whole bunch of products and no way to accept payment for said products? Not very good at all. We're not running a charity (unless, of course, you *are* running a charity). So now we'll look at accepting payment in your new Magento installation.

In this article, we'll be configuring PayPal for payments and walk step by step through the process of getting it set up. Obviously, there are myriad ways you can accept payment, and to discuss them all here would make for a very long article. Covering this option should give you a pretty good idea of the system's basics and allow you to explore alternatives that make sense for your particular situation. Remember, Magento is very flexible and allows you to set up multiple payment options. There really is no limit to how many ways you can take payment in your site.

How to Accept Payments with PayPal in Magento

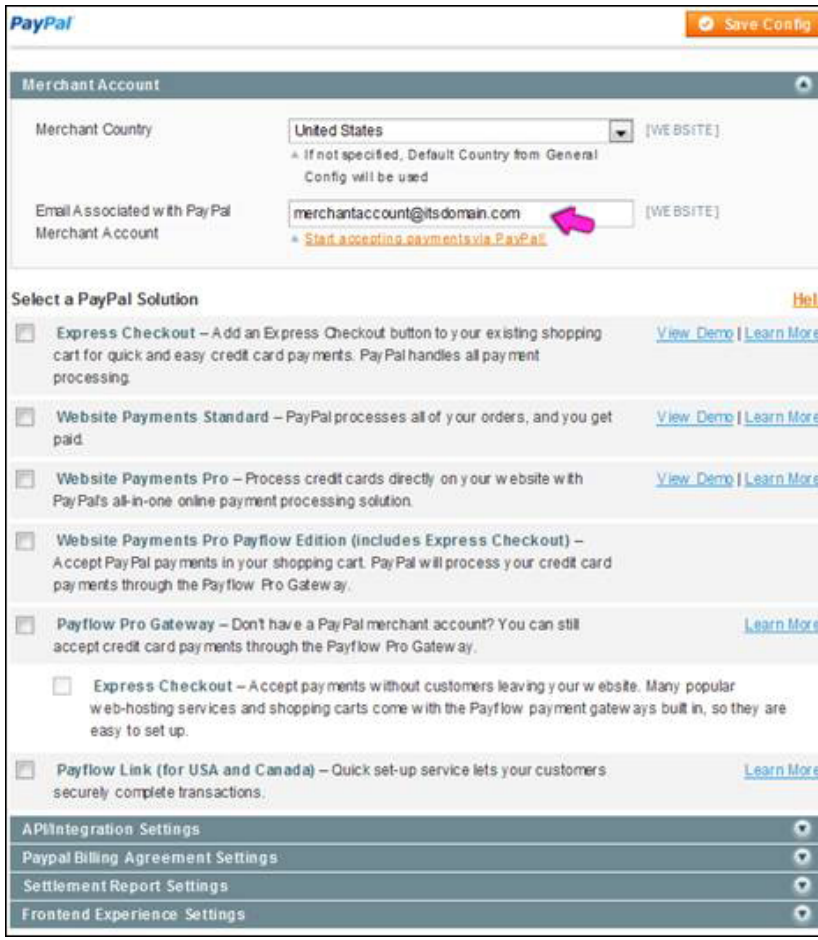
Magento has several ways of integrating PayPal into its system. To configure PayPal, we need to go to the Configuration settings under the System tab from the dashboard of the Magento control panel. From the dashboard, hover your mouse cursor on System and then click Configuration from the drop-down box, as shown below.

You will be directed to the General configuration page. Scroll down the page and look for the PayPal settings link on the left sidebar. It should be under the Sales options.

This opens the main PayPal configuration page. There are several settings you might need, depending on which PayPal method you choose for your particular setup:

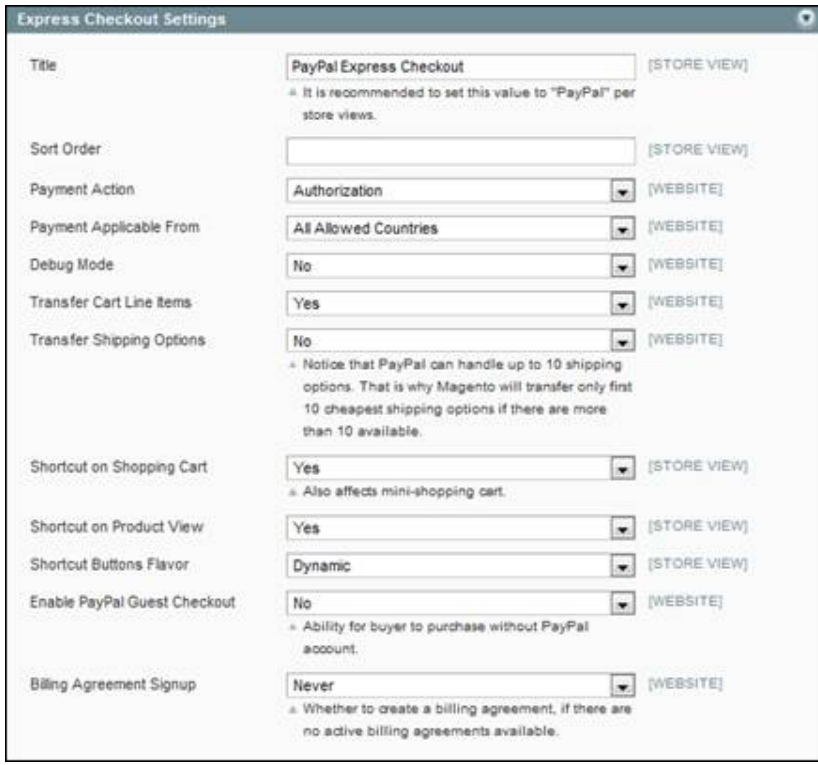
- Express Checkout
- Website Payments Standard
- Website Payments Pro
- Website Payments Pro Payflow Edition (includes Express Checkout)
- Payflow Pro Gateway (with an option to configure Express Checkout)
- Payflow Link (for US and Canada)

Selecting the checkboxes next to each option opens the configuration settings of the various PayPal methods.


PayPal
Save Config
Merchant Account
Merchant Country
United States
[WEBSITE]
If not specified, Default Country from General Config will be used
Email Associated with PayPal Merchant Account
merchantaccount@itsdomain.com
[WEBSITE]
Start accepting payments via PayPal
Select a PayPal Solution
Help
Express Checkout – Add an Express Checkout button to your existing shopping cart for quick and easy credit card payments. PayPal handles all payment processing.
View Demo | Learn More
Website Payments Standard – PayPal processes all of your orders, and you get paid.
View Demo | Learn More
Website Payments Pro – Process credit cards directly on your website with PayPal's all-in-one online payment processing solution.
View Demo | Learn More
Website Payments Pro Payflow Edition (includes Express Checkout) – Accept PayPal payments in your shopping cart. PayPal will process your credit card payments through the Payflow Pro Gateway.
Payflow Pro Gateway – Don't have a PayPal merchant account? You can still accept credit card payments through the Payflow Pro Gateway.
Learn More
Express Checkout – Accept payments without customers leaving your website. Many popular web-hosting services and shopping carts come with the Payflow payment gateways built in, so they are easy to set up.
Payflow Link (for USA and Canada) – Quick set-up service lets your customers securely complete transactions.
Learn More
API Integration Settings
Paypal Billing Agreement Settings
Settlement Report Settings
Frontend Experience Settings

For this article, we are only going to set up the Express Checkout and Website Payments Standard, as they are available to everyone reading this post. When you select Express Checkout, you should see the options shown below. Most of the settings are set to default. Make sure to configure all settings for how your store accepts payments.

Express Checkout – Add an Express Checkout button to your existing shopping cart for quick and easy credit card payments. PayPal handles all payment processing.
View Demo | Learn More



Let's take a look at the various configuration options:

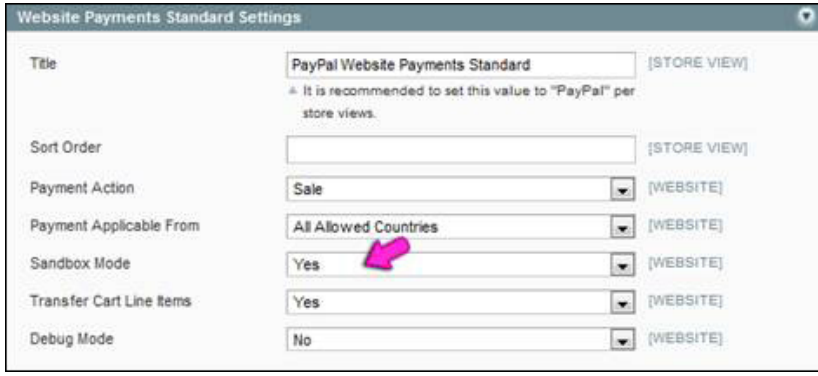
- **Title:** This defaults to PayPal Express Checkout; it is recommended that you leave this configured as is or simply set it as PayPal.
- **Sort Order:** If you are going to configure multiple payment methods for your ecommerce store, this field allows you to enter a numerical value to specify which order the payment methods appear in. So, if you wish for PayPal to appear at the top of the list, enter a 1 in this field.
- **Payment Action:** You have two options here:
    1. **Authorization:** PayPal will only *authorize* the transaction, not actually *process* it. You handle processing in the admin section of the system by creating an invoice.
    2. **Sale:** For most people, this is the option you'll want to select here. This option will actually run the payment through PayPal when an invoice and order are generated.

- **Payment Applicable From:** This option allows you to specify which countries will have this payment option available to them. You may specify only certain countries by Ctrl-clicking or Cmd-clicking on the countries you wish to specify.
- **Debug Mode:** Specify whether you want information written to the log-files about each transaction in the system. Don't worry—credit card information is not written, in order to protect the information. Early on, I'd recommend setting this to Yes; you can always turn it off later if you choose. Be aware that if you have a lot of transactions in your system, it could become unwieldy for you to leave this on, and it may only make sense to use when there is a problem you are trying to diagnose or you recently made a change to the system.
- **Transfer Cart Line Items:** If you set this to Yes, Magento will transfer each item in the cart over to the PayPal website for display. Selecting No will just transfer the cart total.
- **Transfer Shipping Options:** If you're selling digital goods, you can leave this set to No. If you're selling physical products, I would recommend setting this to Yes so that the shipping options are transferred and displayed on the PayPal checkout. Just be aware that only the 10 cheapest options will be transferred to PayPal, as that is the maximum number that PayPal can support.
- **Shortcut on Shopping Cart:** This option indicates whether you want the Express Checkout button available on the shopping cart page. The answer is very likely Yes, so go ahead and make that selection.
- **Shortcut on Product View:** Same basic question here as in the preceding item: do you want the Express Checkout button to show up on the product view? If not, only the Add To cart will be available. This option depends on your situation.
- **Shortcut Button Flavor:** Chocolate or vanilla? OK, not really. Your options here are Static or Dynamic; this affects the PayPal Acceptance Mark image and how it will display next to the payment method on the checkout page. It's recommended that you simply select Dynamic and let the system handle that for you.

- **Enable PayPal Guest Checkout:** This comes down to whether you wish to force registration at PayPal to complete the order. If you select No, users will be able to pay with a credit card without signing up for a PayPal account. If you select Yes, users must either log in with their PayPal account or create one if they don't have an account. Note: If you're selling any items that are subscription-based with recurring payments, you'll need to set this to Yes. Otherwise, you may wish to leave this as No to prevent cart abandonment.
- **Billing Agreement Signup:** Billing agreements are a bit outside the scope of this article. Let's just put it this way: set this to Never if you don't know what a billing agreement is and have never applied with PayPal to have your merchant account enabled with this feature. If you have applied for this with PayPal, then you'll know which option you should choose based on your situation.

Once you've finished configuring PayPal Express Checkout, check the box next to Website Payments Standard, as shown below.

This is the basic PayPal setup for your store. It does not have a lot of options, and we covered most of them above, so I won't rehash them all here. The Sandbox Mode is used when you're testing payments. A Yes value means the store is being used for testing, and No means it is active and will take payments. Because we are using a test store, we will set it to Yes.



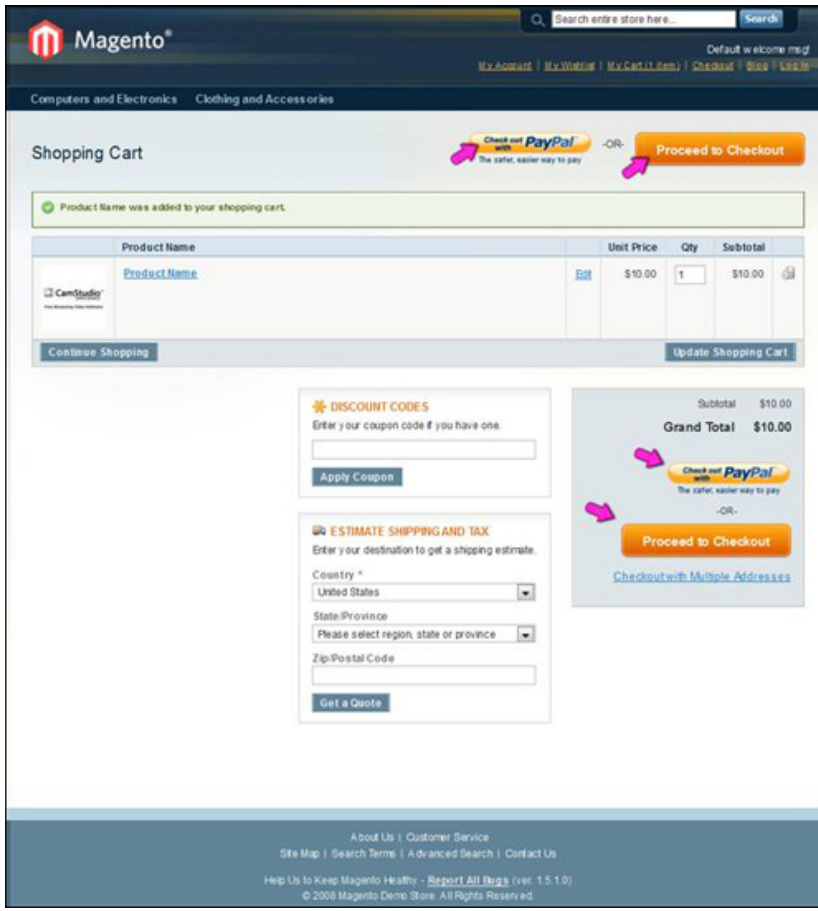
After handling the settings and configuration for Express Checkout and Website Payments Standard, we can now save the configuration from the top of these forms (see the image below).

After saving these settings, we can go to our store to test out the payment method and see how it all looks. The following screenshot displays the list inside the shopping cart. Depending on what you selected in the various settings screens, your view might look a little different. Try adding items to your cart and selecting the "Check out with PayPal" button to make sure that everything works as you expect it to. You'll want to make sure you're in Sandbox Mode at this stage so you're not processing any live orders yet.

Wrapping Up

After completing these steps, as well as those in the previous two articles, you should now have a working store and be able to take payment for items. Fulfillment is a bit of a different beast, and we'll look at some of the ways you can beef up your store's functionality and design in future articles. In the meantime, get your store set up and then drop a link to it in the article's comments section on x.com so we can all take a look. If you have any questions, feel free to add them to the comments section as well so we can help you get your issues resolved.

Also, if you'd like us to cover certain features or items with Magento, let us know. We want to make sure you get the information that's most important to you.



About the Author

Travis Robertson is a peak performance coach and business strategist. He writes about entrepreneurship, leadership, and millennials on his blog, TravisRobertson.com. You can follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 14

Magento, Part 4: Integrating Magento and WordPress Using the FishPig Extension

Travis Robertson

This article is part of a larger series on the Magento ecommerce platform. In this post, we're going to look at how to integrate Magento with WordPress using the FishPig extension for Magento.

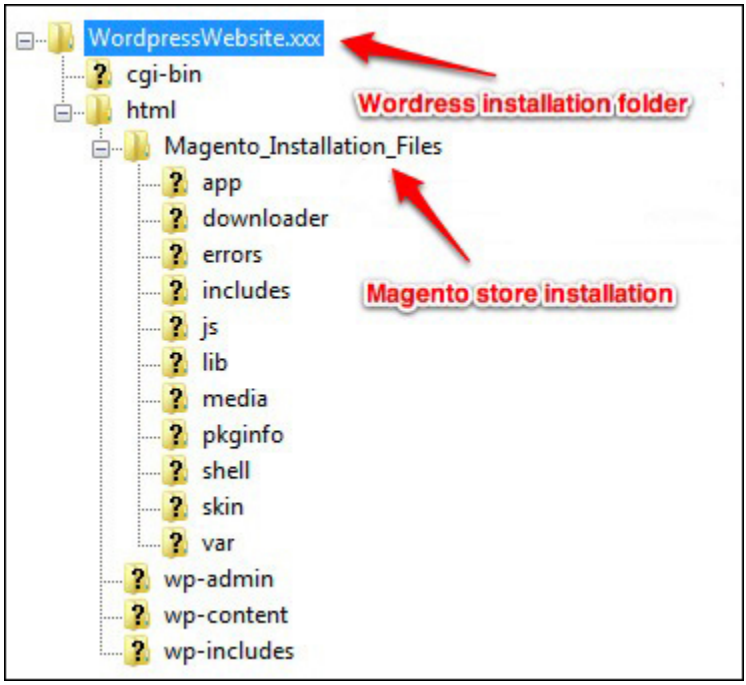
In Part 1 of this series, I took you step by step through the installation and setup of Magento. I pointed out some of the gotchas along the way and the various configuration options available to you. When we were done, we had a working (albeit a rather plain) ecommerce site set up.

In Part 2, we looked at getting products into your new Magento store. We discussed how Magento organizes products and inventory into categories and subcategories, and covered some of the potential configuration options. I also gave you a homework assignment to get your products into your new system so that we could begin the process of accepting payment.

In Part 3, we walked through setting up PayPal in your store so that you could accept payment for your products. We also looked at some of the configuration options that allow you to place Buy Now buttons in various locations of the store based on your needs.

Integrating Magento and WordPress with FishPig

Before starting with the steps provided here, you should already have Magento and WordPress installed on your system, and you should have basic connection information for the databases you will be connecting to. If you're creating clean installs of both systems, you can use the same database for both WordPress and Magento. You may also choose to create a separate database dedicated to your Magento store. For this tutorial, we installed the Magento files beneath our WordPress installation folder. The screenshot below provides an overview of the complete file structure.

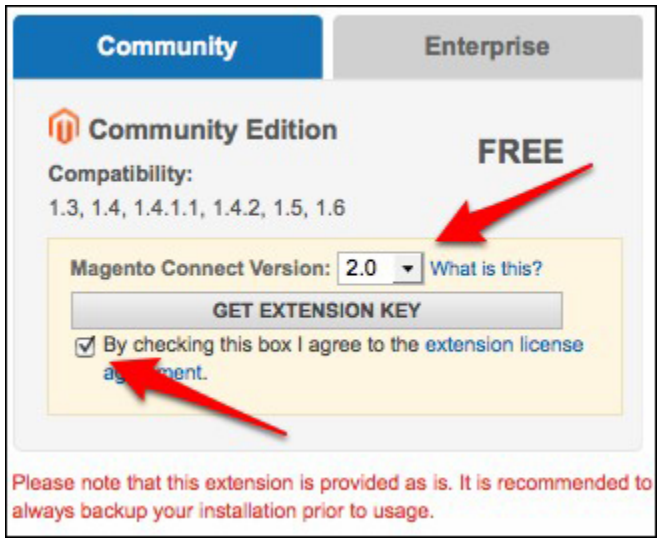


After getting your system ready, go to the Magento Extensions pages and look for the FishPig extension. You'll need an extension key, and in order to get that key, you will need to register as a member of the Magento community. Registration is free; if you are already a member, make sure you are logged in to retrieve the extension key.

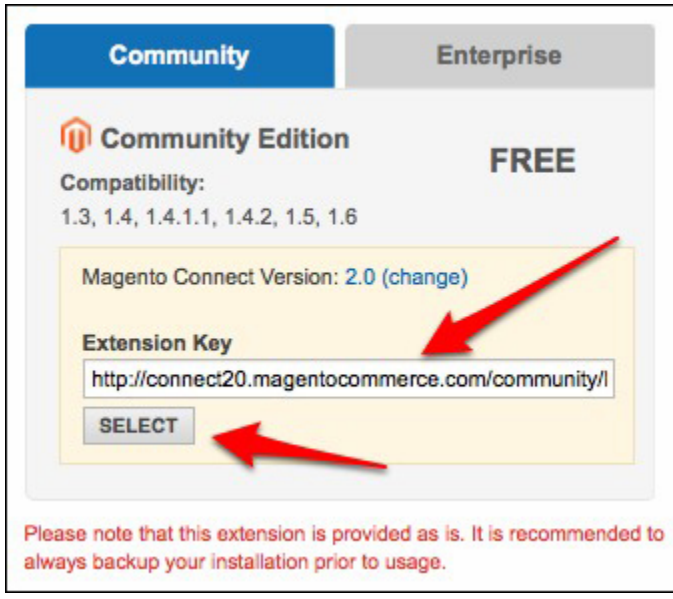
As noted in the following example, it is important to back up your installation before using this extension. In fact, always make a backup of your current working system before making any changes.

From the FishPig Extension page, click the Get Extension Key button. You will be prompted next to specify which version of Magento Connect you have. If you are not sure which version you have, you can check it here. The Magento system we are using for this illustration is version 1.5.1.0 Community Edition, which is supported by Magento Connect version 2.0.

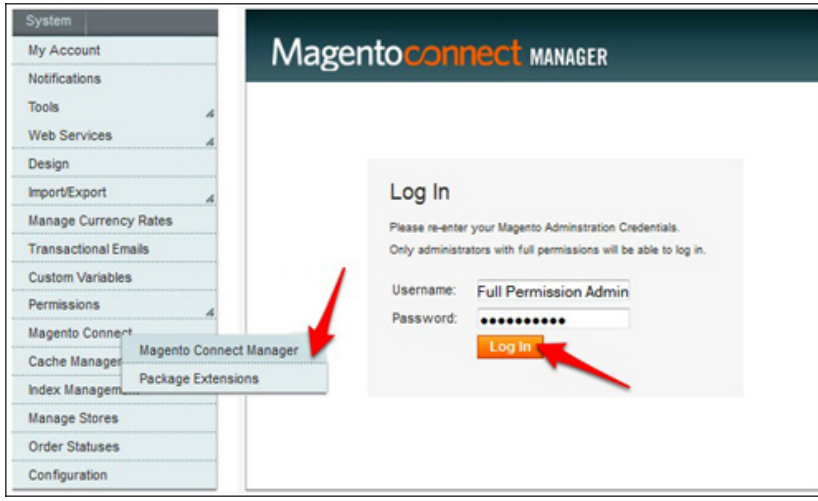
Community
Enterprise
Community Edition
FREE
Compatibility:
1.3, 1.4, 1.4.1.1, 1.4.2, 1.5, 1.6
Get Extension Key
Please note that this extension is provided as is. It is recommended to always backup your installation prior to usage.


Community
Enterprise
Community Edition
FREE
Compatibility:
1.3, 1.4, 1.4.1.1, 1.4.2, 1.5, 1.6
Magento Connect Version: 2.0
What is this?
GET EXTENSION KEY
By checking this box I agree to the extension license
Please note that this extension is provided as is. It is recommended to always backup your installation prior to usage.

After selecting the Magento Connect version we will be using, select the box labeled "By checking this box I agree to the extension license agreement." and then click the Get Extension Key button. When the extension key appears, click Select to highlight it. Right-click on the highlighted address, select Copy to retrieve the extension key, and then paste it in Notepad or another text editor to keep it safe.

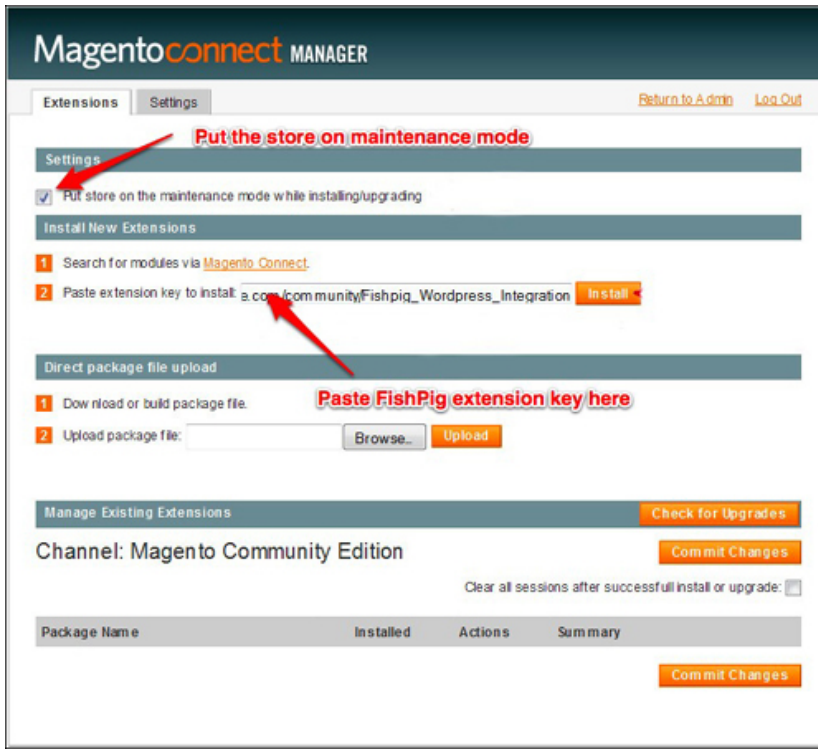


After getting our extension key, the next step is to log in to our Magento dashboard. From the main navigation, hover your mouse cursor on the System link. From the drop-down menu, hover over Magento Connect, and then click Magento Connect Manager. You will be prompted to enter your username and password again; since you will be making a big change to the actual system, Magento needs to know if your account has the required permissions to do so. If you don't, you will not be able to make the necessary changes and will need to get an account with full permissions.



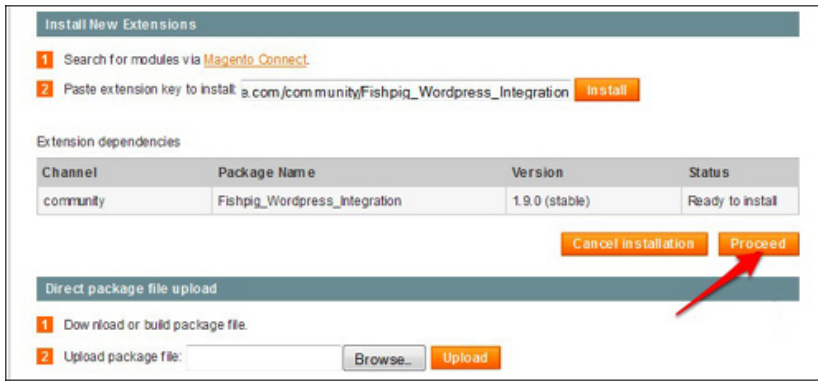
After successfully logging in, you will be shown the Magento Connect Manager page. As you can see from the following illustration, the interface is quite simple. On the Extensions installation page, before doing anything else, it is important to select the "Put store on the maintenance mode while installing/upgrading" option. This is the first thing you should always do, especially if you are working on a live store or website. Although the FishPig extension does not modify Magento's core files, configuring the settings between your WordPress and Magento files may cause errors if, by some chance, you have a customer who is about to purchase something on your store from the frontend. By checking this option, you will hold off all transactions on your store for the duration of the installation, which will probably take just a minute or two.

The second step of this procedure is to copy and paste the extension key (actually, a URL of the extension's installation file) in the field provided. After pasting, click the Install button. Don't worry; it will not install the files just yet. First, the system will check and retrieve information about the extension.

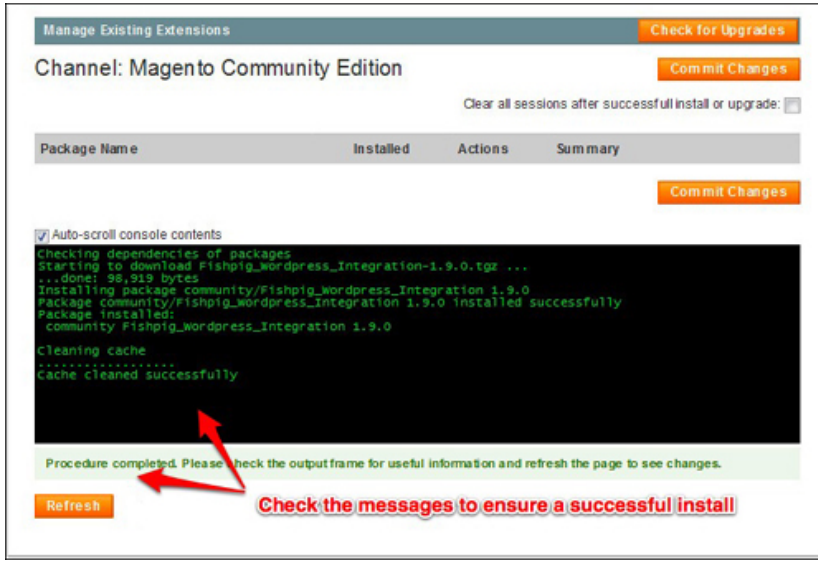


If the Magento system successfully retrieves the information about the FishPig extension, a small window appears, displaying the Channel, Package Name, Version, and Status. Check to see if the information matches the extension you are installing. This information is important to verify before you click Proceed. For our discussion, we are trying to install the FishPig's WordPress/Magento Integration, using a Community Edition Magento system, with a stable 1.9.0 version, and a status of "Ready to install."

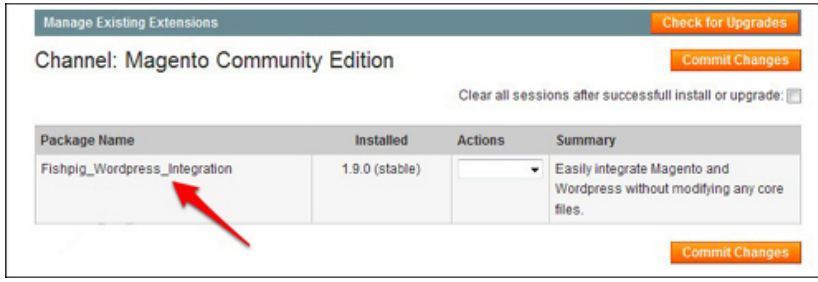
Given the fact that we are integrating two different systems, I recommend you use only stable (non-beta and non–release candidate) versions of the FishPig extension. If you see beta or release candidate versions, avoid upgrading or installing these extensions because they likely still have bugs and errors. However, if all the information is correct and it looks like the illustration below, we can continue by clicking Proceed. Note that now is the time when the actual installation process occurs.



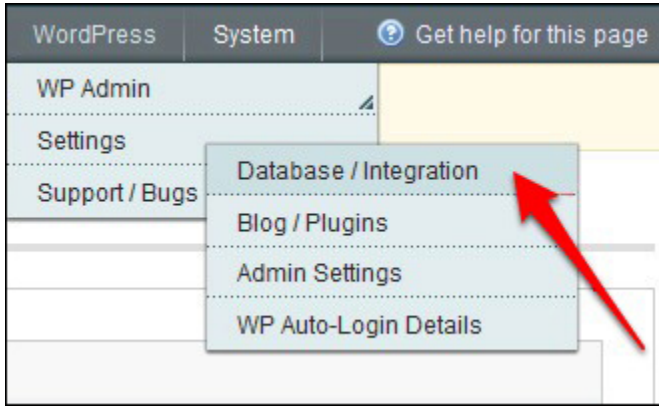
It can take anywhere from a few seconds to a few minutes to install FishPig's WordPress Installation package. As you can see from the image below, a console displays the installation progress. If the process is successful, it should display a message below the console indicating that the process is complete. When you see this, click Refresh to see if this extension appears on the list.



After refreshing the page, you should be able to see the FishPig extension listed under the packages installed in your Magento system. That completes the installation process of the FishPig extension's files. The next step involves connecting your WordPress installation and your Magento store by configuring settings on both the Magento and WordPress dashboards.

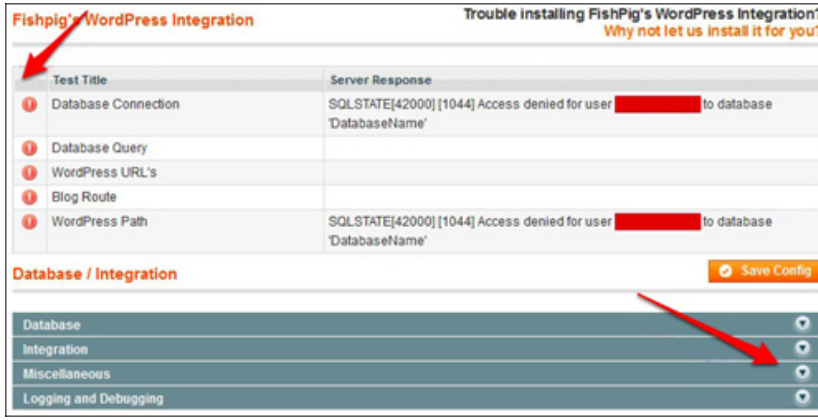


After a successful installation of FishPig's WordPress Integration extension, log out of the Connect Manager and go back to your Magento dashboard. You should see a new link from the dashboard, labeled WordPress. The options here require us to fill in the neccessary information to properly configure WordPress and Magento together. Hover your cursor on the links and drop-down menus, and click Database/Integration.



We are now on the page where we can connect the databases of our Magento store and our WordPress website. Before we continue, bear in mind how our databases are set up. We should know how we have set up WordPress and Magento during their respective installations. In our case, the two systems were installed independently of each other, with each system having its own database. This small piece of information is important for us to know before going ahead with the actual integration process.

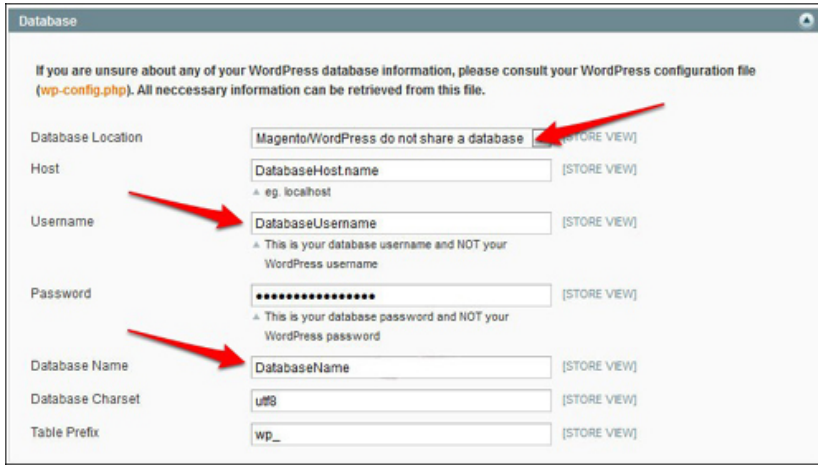
As shown below, we have Magento, WordPress, and the FishPig integration extension all working, but we need to configure them. On the left, we can see small red exclamation points, which indicate that they need to be configured properly. We will need to fulfill their respective requirements in order to properly connect our systems. You can change these settings through the green tabs below the forms. Click on the small circular arrows to open each connection setting.



First is Database. Remember that, in our example, we have separate databases for WordPress and Magento. Therefore, we will select "Magento/WordPress do not share a database."

Database
If you are unsure about any of your WordPress database information, please consult your WordPress configuration file (wp-config.php). All neccessary information can be retrieved from this file.
Database Location
Magento/WordPress share a database
Table Prefix
Magento/WordPress share a database
Magento/WordPress do not share a database
[STORE VIEW]

Next, new options for our database will appear, and we need to fill in all required fields for it to work. To input the information below, you need to know your WordPress installation information: the host, database username, database password, and the database name. Remember that we are speaking of the WordPress *database* information, not the WordPress *login* information. Depending on your web hosting company, you may need to log in to your hosting administration site to retrieve this information.



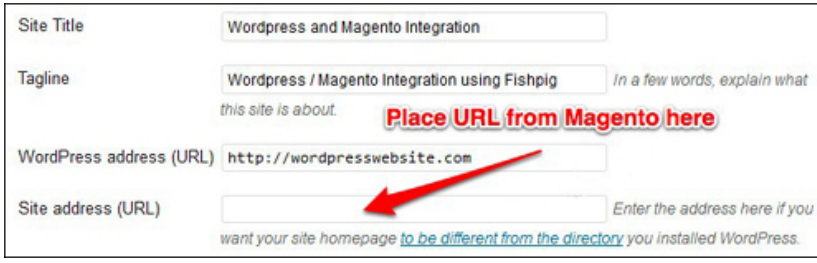
After filling in the correct information, click on the Save Config button. If all the information is correct, the small red exclamation point should turn into a small green checkmark, and you'll see a smiley face :) on the right.

As shown below, the database information we entered was correct, so the next step is now to follow the instructions provided by Blog Route and WordPress Path. For Blog Route, copy the URL provided on the right and log in to your WordPress dashboard.

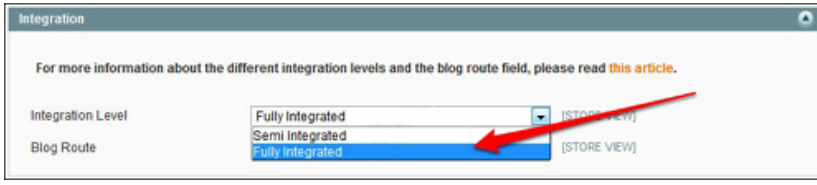
Test Title
Server Response
Database Connection
:)
Database Query
:)
What you place here will be added to WP
WordPress URL's
:)
Blog Route
Go to the General Settings page of your WordPress Admin and set the 'Site address (URL)' field to 'http://
WordPress Path
Your WordPress path is incorrect. You must set the path to your WordPress installation
Database / Integration
Save Config

From our WordPress dashboard, go to the Settings tab and click General.

This is a part of the General Settings page in our WordPress dashboard. Paste the URL we got from Magento into the "Site address (URL)" field, and then save the settings by clicking the button labeled Save Changes.

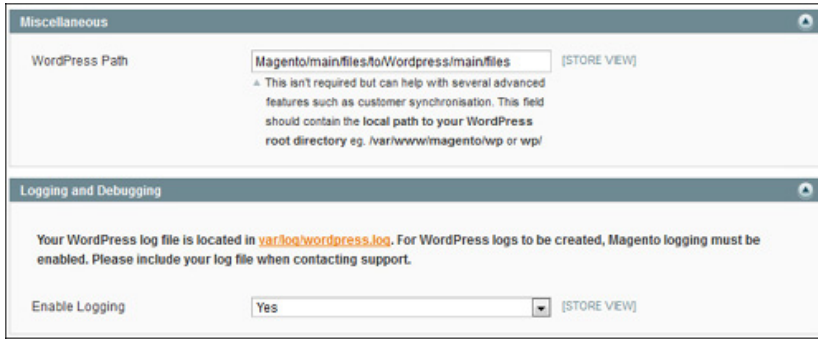

Site Title
Wordpress and Magento Integration
Tagline
Wordpress / Magento Integration using Fishpig
In a few words, explain what this site is about.
Place URL from Magento here
WordPress address (URL)
http://wordpresswebsite.com
Site address (URL)
Enter the address here if you want your site homepage to be different from the directory you installed WordPress.

We can now go back to Magento. From the Integration tab, we will select Fully Integrated.

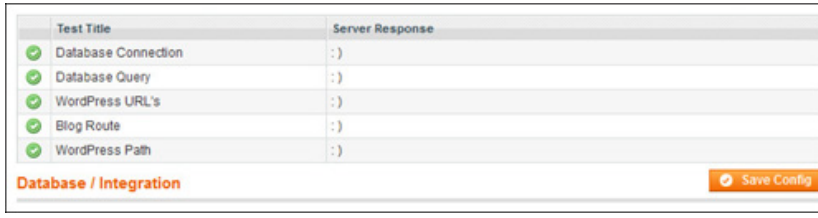


On the Miscellaneous tab, type in the WordPress path. The path root is the Magento Installation folder and leads to the WordPress installation's *index.php* file. Based on the file structure shown at the beginning of this discussion, our WordPress path is (../)

The last part is to make sure that Enable Logging is set to Yes from the "Logging and Debugging" tab. This is important to trace errors and logs in any exchange from your Magento and WordPress systems.


Miscellaneous
WordPress Path
Magento/main/files/to/Wordpress/main/files
[STORE VIEW]
This isn't required but can help with several advanced features such as customer synchronisation. This field should contain the local path to your WordPress root directory eg. /var/www/magento/wp or wp/
Logging and Debugging
Your WordPress log file is located in var/log/wordpress.log. For WordPress logs to be created, Magento logging must be enabled. Please include your log file when contacting support.
Enable Logging
Yes
[STORE VIEW]

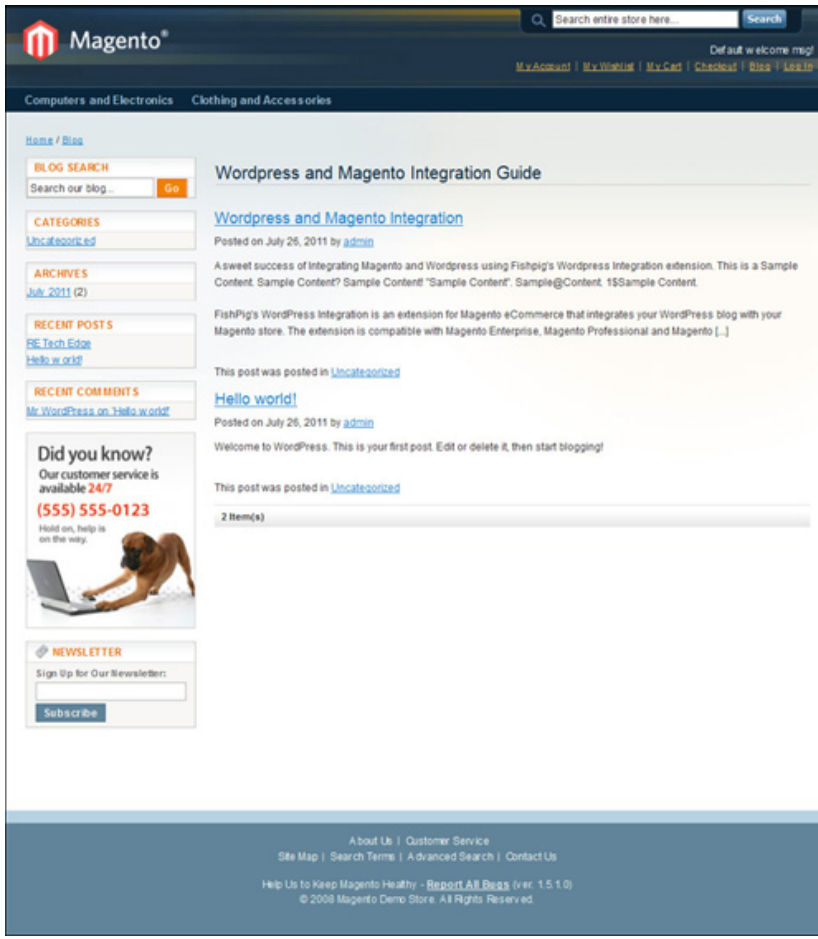
After this, click Save Config again to enable the changes we made. If all goes well, none of the tiny red exclamation points will appear—just the little green checkmarks and smileys that confirm the success of our tests.



We can verify the integration from the frontend of our store with the URL that we used in our WordPress General Settings. Simply navigate to that URL in your browser, and you should be able to open your Magento store with your WordPress content in it.

Wrapping Up

If you have any questions about your installation, drop them in this article's comments section on x.com . Also, if you have a site that runs on WordPress and Magento and would like to share it with the community, add that to the comments section as well.


Magento
Search entire store here...
Search
Default welcome msg!
My Account | My Wishlist | My Cart | Checkout | Blog | Log In
Computers and Electronics
Clothing and Accessories
Home / Blog
BLOG SEARCH
Search our blog...
Go
CATEGORIES
Uncategorized
ARCHIVES
July 2011 (2)
RECENT POSTS
RE Tech Edge
Hello w orld!
RECENT COMMENTS
Mr WordPress on Hello w orld!
Did you know?
Our customer service is available 24/7
(555) 555-0123
Hold on, help is on the way.
NEWSLETTER
Sign Up for Our Newsletter:
Subscribe
Wordpress and Magento Integration Guide
Wordpress and Magento Integration
Posted on July 26, 2011 by admin
A sweet success of Integrating Magento and Wordpress using Fishpig's Wordpress Integration extension. This is a Sample Content. Sample Content? Sample Content! "Sample Content". Sample@Content. 1$Sample Content.
FishPig's WordPress Integration is an extension for Magento eCommerce that integrates your WordPress blog with your Magento store. The extension is compatible with Magento Enterprise, Magento Professional and Magento [...]
This post was posted in Uncategorized
Hello world!
Posted on July 26, 2011 by admin
Welcome to WordPress. This is your first post. Edit or delete it, then start blogging!
This post was posted in Uncategorized
2 Item(s)
About Us | Customer Service
Site Map | Search Terms | Advanced Search | Contact Us
Help Us to Keep Magento Healthy - Report All Bugs (ver. 1.5.1.0)
© 2008 Magento Demo Store. All Rights Reserved.

About the Author

Travis Robertson is a peak performance coach and business strategist. He writes about entrepreneurship, leadership, and millennials on his blog, TravisRobertson.com. You can follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 15

The Convergence of Local, Social, and Mobile, Part 1: Local

Travis Robertson

Have you ever had one of those "Ah ha!" moments? You know the ones: you're going about your day, doing whatever it is you normally do, and then *bam*! It hits you. Hard. You weren't expecting it. Nothing about the day seemed unusual. You didn't wake up thinking to yourself, "I feel like I'm about to get hit with a lead pipe of insight today."

But then it happens. And, all of sudden, you can't help but look at the world—your world—through the lens of your newfound inspiration. You begin to see new opportunities popping up all around. It's like springtime in your mind.

Now, lest you think I've smoked some sort of illicit drug, let me explain myself.

Back in October, I was watching a video (*http://ustre.am/5UOE*) from the PayPal X Innovate 2010 Conference to prepare for an article I was writing. It's a keynote speech from Osama Bedier, PayPal's former Vice President of PayPal Platform and Emerging Technology. About twelve and a half minutes in, Bedier made the point that prompted my "moment." And it was fleeting—for Bedier, the point was simply intended to preview a few other ideas before he moved on. But it's continued to stick with me.

So much so that when Matthew Russell approached me about doing an article series for X.com, I asked him if I could write a four-part series on this topic I found so fascinating.

What is this incredible, life-changing concept?

The convergence of mobile, local, and social, and how it will ultimately impact our lives as the lines between these areas continue to blur.

None of these three areas is life-changing on its own. We've been living with some form of them for millennia (or decades, in the case of mobile). But their convergence is setting the stage for a new form of interaction, and that's what I'll discuss in this four-part series.

Breaking Silos

To fully appreciate this convergence, we must first understand the siloed nature of the three elements. For those of us who are younger, we can sometimes take this for granted since we have little to no appreciation for how the world used to work.

For those who are a bit older, perhaps you can remember a time before mobile phones. You've seen all of the changes and, while you can appreciate them, you haven't really stopped to think about just how far we've come and what these changes really imply about where we're headed. Not to mention, very few people alive today (and, I would argue, not a single person reading this) can remember a time without cars.

That is why I want to start by looking at how mobile, social, and local have evolved. We'll tackle each in a different article. Then I'll wrap it all up by looking at their convergence and how this will impact what we can expect moving forward.

I will avoid any "flying car"–type predictions. It's all too easy to forecast out so far that we're dealing in the fantastical. I don't want to do that. I want to examine the real, the likely, and the possible. I'll let the science-fiction writers take it beyond that.

Defining Local

Local and social often seem so intertwined that we view them as the same concepts. After all, for millennia, social was only possible locally. But for the sake of this series, it's important to separate them.

Local is a geographical designation. *Social* deals with how we interact or engage with one another. Keep this distinction in mind as we look at the local silo. I know it seems kind of pedantic to define a term such as "local," but I had to grapple with the differences between local and social as I was preparing to write this, so I thought it would be helpful to you to understand them as well.

The Evolution of Local

It's a strange exercise to contemplate the evolution of the concept of "local." We tend to define local based on our own experiences. Let me give you a few examples:

- I live in Franklin, a city about 20 miles south of Nashville, TN. Both Franklin and Nashville are "local" to me, as are about 15–20 other cities and towns—maybe more. I tend to define "local" as within a 90-minute drive. That's odd for this part of the country, but I grew up in Los Angeles, where 90 minutes was my average drive time to just about anywhere in Los Angeles County.
- My grandparents lived on a 165-acre ranch in Northern California. The nearest town was about an hour away by car. To them, this was local. In fact, it was the only town within a couple of hours.
- In some countries, "local" is the village you live in because your only mode of transportation is your own two feet.
- In Hawaii, it's not uncommon to commute to work using a 25-minute plane ride to another island.
- In New York, some people commute to work from surrounding states using trains, while others walk a couple of blocks. The two groups view "local" differently.

Do you see the conundrum? Local is relative to our experiences and expectations. But here's the interesting thing: it's only more recently become a challenge to define.

Phase 1: Pre–personal transportation and technology

If you're an historian, you're probably going to hate me when I'm done with this article. (Sorry.) But for the sake of simplicity, I have to make certain delineations and leave out large amounts of nuance and data. These phases are by no means perfect or complete. However, I think they are helpful in promoting a common understanding.

The first understanding of "local" centered on the concept of "immediacy." Your immediate community. Your immediate village. Your immediate family. Your immediate circle of friends and acquaintances. "Locally grown" meant you grew it in your backyard. "Local news" meant village or town gossip. Without transportation to go beyond where your legs could take you, the world was quite small.

Local was an idea constrained by both distance and time. Even hopping on a horse or riding in a carriage didn't do too much to change this on a global scale. While distance and time were important, the emphasis was arguably more about comfort of travel than speed of travel. Caravans of horses, people, and carriages still had to move at a walking pace. They simply provided companionship and security to those making the journey.

On the whole, most transportation improvements up until about the mid-1600s focused on the experience or utility of traveling. One exception was maritime travel, which could oftentimes allow for more direct routes to destinations across bodies of water.

However, despite all of the improvements in travel over thousands of years, little was done to change the idea of "locality." Why? Because transportation wasn't really focused so much on the personal as it was on the utilitarian. Innovations were often made for military reasons. The average person didn't really travel unless migrating.

At the end of the day, local was what was immediately accessible to you. (Somewhere, there is an historian who wants to punch me right now. Please stay with me.)

Phase 2: Introduction of personal transportation

Again, with the exception of maritime travel and (arguably) caravans, mass transportation wasn't practical until around the mid-1600s, when Blaise Pascal (yes, the same guy who gave us the calculator and got a programming language named after him) invented the first public bus, which was horse-drawn and included a fare system, in Paris. While it probably didn't change the world overnight, it planted a concept in people's minds that would continue to grow: public transportation.

Humankind then saw the rise of steam engines in the 1700s, followed by their use for transportation in the 1800s. Steam-powered boats and locomotives allowed for expansion beyond major city centers. I love the prediction below from Oliver Evans—the man credited with perfecting the steam engine:

> The time will come when people will travel in stages moved by steam engines from one city to another, almost as fast as birds can fly, 15 or 20 miles an hour.... A carriage will start from Washington in the morning, the passengers will breakfast at Baltimore, dine at Philadelphia, and sup in New York the same day.... Engines will drive boats 10 or 12 miles an hour, and there will be hundreds of steamers running on the Mississippi, as predicted years ago.
>
> —Oliver Evans, 1800

Explorative travel became easier, and increasing numbers of people had access to systems that would allow them to travel back and forth more readily between remote destinations is much shorter periods of time.

I believe this is where the concept of "local" began to expand a bit. News traveled much faster (thanks in large part to the telegraph, which we'll cover in another article). The flow of news, information, products, services, trade, and so on, all moved with increasing speed. The world was shrinking. Remote locations didn't feel as remote.

Then, in the 1900s, the ultimate form of personal travel hit the scene: the automobile. The love affair with the car happened in large part because of the freedom it afforded the average person. Suburbs sprung up, and commuting to work became the norm. "Local" was being redefined.

Whereas local used to be constrained by both distance and time, it was now shaking free of the bonds of distance. After all, 20 minutes by foot would get you about a mile if you walked briskly. Now, by car, it could get you to work 15 miles away.

Local changed. What used to be about immediacy now became about *reasonable access*. Sure, "reasonable" is relative. To those who had cars, it was reasonable to drive 30 miles. To those without cars, 30 miles was a 10- or 12-hour journey and not reasonable. However, this was really the first time the definition of "reasonable" was debatable.

Phase 3: The information "superhighway"

I really hate the term "superhighway." It sounds so dated. However, it's relevant to a discussion about how the concept of local has changed.

What I'm really referring to is *networked technologies*. The telephone made it possible to talk to someone miles away in ways that were previously impossible. Although telegraphs and handwritten letters existed, there was something about hearing someone else's voice that changed how we communicated. It took a bit of the sting away from distance.

Fast-forward to 1989, when a scientist by the name of Tim Berners-Lee invented what we now know as the World Wide Web. This technology would go on to forever change our concept of local. Let me explain.

As the Internet grew in ubiquity and we realized how easily and quickly data could be sent by the average person with a computer to someone on the other side of the world, we placed a huge emphasis on the "remote." Location and physicality seemed less important. We outsourced. We hired remote workers. We telecommuted.

Next, as the bandwidth of lines into homes increased, video came on the scene. We could talk "face to face" with family members in other parts of the world. Distance lost even more of its sting. Local became less important. After all, we could accomplish everything we needed to using web-based tools.

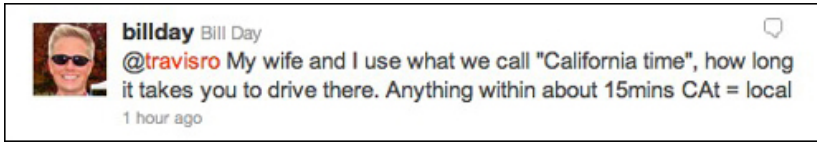
As the world shrank, we were now in a phase where time and space no longer seemed to matter. The only thing the Internet seemed incapable of was providing physical interaction. Beyond that, it seemed we no longer desired the local. We even shopped online at sites like eBay (the parent company of this site) and Amazon.

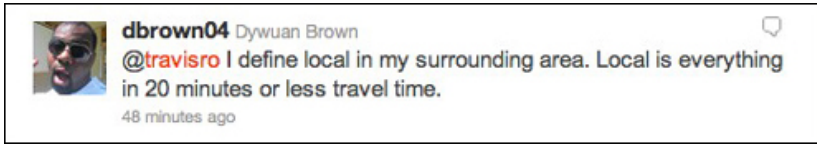
Then something happened. But you'll have to keep reading this series to find out what it was.

Share Your Thoughts

How do you define local? Do you use time, space, physicality, a combination of those, or something different? Share your thoughts in the article's comments section on x.com. (Note: You must be logged in to leave a comment.)

To get you started, here are some of the responses I got from people when I put that question out on Twitter. How does your definition line up?


billday Bill Day
@travisro My wife and I use what we call "California time", how long it takes you to drive there. Anything within about 15mins CAt = local
1 hour ago


dbrown04 Dywuan Brown
@travisro I define local in my surrounding area. Local is everything in 20 minutes or less travel time.
48 minutes ago

kennysilva Kenny Silva
@travisro Local, for me, extends about 30 minutes from the center. Nashville,Franklin,Hendersonville,Bellevue,Mt. Juliet.
1 hour ago
MaxWeb Jim
@travisro Local is here in our town. Local is not Million Dollar Franchise or Big Box, it's Mom & Pop & the people who can't live w/o your $
1 hour ago
MKMartin Matt Martin
@travisro For me, local is my region, Middle Tennessee. I consider regional farmers as locals.
1 hour ago
QuietRumbling D. A. Schweiss
@travisro Good question! I define local by what I can reach within a 15 minute drive. Beyond that, it's sub-local and then long-distance.
1 hour ago
TonjaC Tonja Conway
@travisro We have a cluster of cities along the highway, approx 20 mi radius. I'd consider that local. Might vary if I lived elsewhere.
2 hours ago
VillageAdsSeo Tim Biden
@travisro That depends on the application. Shopping, friends, family, parks, hiking trails? They all have different definitions.
1 hour ago

About the Author

Travis Robertson is a business strategist and consultant. He also writes about entrepreneurship and leadership and millennials on his blog TravisRobertson.com. Read more from him on his X.com blog, or follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 16

The Convergence of Local, Social, and Mobile, Part 2: Social

Travis Robertson

This is Part Two in a four-part series on the convergence of local, social, and mobile. If you have not read the first part of this series, you might want to start there; I reference some things throughout this piece that won't make sense without it.

For those who don't really feel like being caught up, I'll recap briefly.

Back in October, I was watching a video (*http://ustre.am/5UOE*) from the PayPal X Innovate 2010 Conference to prepare for an article I was writing. It's a keynote speech from Osama Bedier, PayPal's former Vice President of PayPal Platform and Emerging Technology. About twelve and a half minutes in, Bedier made the point that prompted my "moment." And it was fleeting—for Bedier, the point was simply intended to preview a few other ideas before he moved on. But it's continued to stick with me.

As a result, I wanted to write a series for the PayPal X developer portal covering the convergence of local, social, and mobile, and how it will ultimately shape our more immediate future. And for those who did not read the first article, the future I am going to cover is not the "flying cars"–style future, where I predict so far out it becomes ridiculous. Instead, I'll focus on 1–2 years out.

But, for us to properly understand the future, we must first understand our past. So this article will cover the evolution of "social" and how we have arrived at our current view on the topic.

Briefly Defining Social

As I mentioned in the first article, the concepts of "local" and "social" often seemed so connected that we have difficulty trying to separate them and evaluate one without the other. However, I think it's critical that we distinguish between the two so that we can fully appreciate their convergence.

For the purposes of this series, *local* is a geographical designation. *Social* is how we connect with and engage with one another. Understanding that will help you as we examine the social silo.

The Evolution of Social

In many ways, I think we have done a disservice to the word *social* in recent years. The rise of *social media* and *social networking* as buzz phrases seems to imply that these concepts never existed prior to the Internet. For those who are Millennials (born between 1977 and 1995, approximately), it would be easy to assume from the way in which "social" precedes "networking" and "media" that neither of those two things had any social component until recent years.

Perhaps people would congregate at a bar, night club, restaurant, or conference, but never really socialize. Those poor souls without Twitter and Facebook. How did they survive without any social component to those gatherings? Thanks be to Zuckerberg and @ev for introducing social into our lives!

The reality is that networking is a social activity by definition. In fact, the term *social networking* is almost redundant (if not fully redundant). And media—while certainly not inclusive of "thumbs up," "like," or "stumbling"—has always contained a social component.

That's what I want to look at in this article. I want to get to the heart of how "social" has evolved to include thumbs, likes, tweets, and Diggs.

Phase 1: "Can I borrow your goat?"

Now, obviously I have no clue as to whether people used to borrow one another's goats. However, what I do know is that early social interaction was predicated on language. Additionally, it was constrained by locality. Only those whom you could communicate with both in person and locally were really considered to be a part of your "social network." (See? Contrary to popular belief—thanks in part to a movie by the same name—social networks were not invented by Mark Zuckerberg. Just saying.)

The social network usually consisted of a village, tribe, or family with an occasional neighboring village, tribe, or family thrown in to prevent a really small gene pool. However, at this point in history, communication required at least two people who were geographically bound together meeting face to face. Their version of a text message was sending the fastest member of the clan on a run to the recipient's village—not a fun job if you were carrying bad news like a declaration of war or a long-distance breakup. Chances were good that being the bearer of bad news would get you killed. It's where we get the phrase "don't kill the messenger."

Phase 2: Pen pals, trade, and travel

Around 4,000 BC, people got tired of trying to remember everything that was said to them (or by them). Cities were growing bigger, trade was becoming more important, and accurate record keeping was becoming more necessary. It's not really known what the first writings were. There are speculations that writing started with cave walls. Others think it might have been rocks, stone tablets, or clay. For the sake of this post, it does not matter. What matters is how the advent of writing, coupled with the expansion of both the methods and the speed of travel, created opportunities for interacting beyond small geographic regions. Ideas could spread more quickly. Rulers, philosophers, religious and political leaders, and the educated could send messages to far reaches of the globe by combining the social component of writing with the delocalization provided by various methods of travel.

And while there were definite drawbacks and limitations inherent in such a system, it allowed for the world's first major superpower to emerge: Egypt. It also enabled the educated to communicate and socialize despite their lack of proximity to one another.

Don't underestimate the significance of this. Without writing, delocalization would have been nearly (if not fully) impossible. "Social" would have been dependent on physical locality. Only if you traveled could you have friends or acquaintances in geographically remote locations. Additionally, your only ability to interact with them would have been if you were standing face to face.

Instead, those who did travel to various regions had the capacity to maintain a social connection by interacting and engaging with those they met through the written word. World religions such as Judaism, Christianity, and Islam, along with philosophies such as those of Socrates and Plato, were spread because people could travel and communicate via letters to other adherents. That "social" connection expanded many people's "social network." Even couch surfing isn't a new idea: many people sent letters ahead to friends or family in other regions asking them to provide shelter for a traveling friend.

For many years, writing and travel were the two primary methods for expanding your social network. While undoubtedly beneficial, they were still limited by time. It could take days, weeks, or months for letters to arrive, depending on the distance one had to travel and the conditions of the journey. Additionally, there were little to no guarantees that the letters would even make it to their destinations.

Phase 3: The rise of instant communication

The Pony Express got a raw deal with the rise of the electronic telegraph. In the United States, the Pony Express was a big deal, as it was responsible for transporting communications between coasts.

However, between 1837 and 1838, two independently developed electronic telegraphs were invented (one in the United Kingdom by Sir William Fothergill Cooke and Charles Wheatstone, and the other in the United States by Samuel Morse). [Fun fact: Ancient versions of the telegraph existed, using smoke signals and reflected light. However, they couldn't travel very far and depended heavily on weather.] The telegraph provided a method for instant communication across large land masses and, as of 1866, across bodies of water such as oceans.

After the telegraph came the telephone and then the radio. [Note: The historical order of these two is dependent on which events you use to determine their dates of invention and use. However, they were created and used in close proximity to each other.] These devices allowed for instant communication, the telephone for one-to-one communication and the radio for one-to-many communication (technically, a broadcast, since a listener is incapable of communicating back to the broadcaster).

Finally, instant communication has brought us to our current state, which results from the invention of the Internet. Whereas we were once only able to transmit individual messages between a sender and a receiver or use a broadcast system with no interaction, thanks to the Internet, we now have the capacity to communicate simultaneously with multiple recipients in real time to and from anywhere in the world. Our social networks are now empowered with the speed of instant communication and the almost complete irrelevance of geography.

Social—a concept once limited due to time, space, and language constraints—is now possible regardless of those three factors. Social networks are nothing new to mankind. Instead, our understanding of them as sites such as Facebook or Twitter is simply the latest evolution in something that has been occurring since the dawn of time.

A Final Note on Social Media

I alluded to the fact that media has always been social, and I don't want to finish this post without addressing that. According to Wikipedia, *media* is simply "the storage and transmission channels or tools used to store and deliver information or data." That's it. It's dependent on communication or "social" interaction:

- When stories were told around village fires as a way of keeping oral traditions alive, that was both social and a form of media.
- When plays were created as another method of conveying a message, they required an audience and were often discussed, dissected, and critiqued. Plays were (and are) both social and media.
- When books were written, they too were critiqued, reviewed, and rebutted. Again, they are both social and media.
- The same is true of radio, newspapers, television, and movies.

Just like social networking, *social media* is far from a new concept. Our current forms—such as YouTube, Digg, StumbleUpon, Delicious, blogs, and the like—are simply the evolution of things that already existed in various formats. Now, instead of gathering at the water cooler to discuss the latest episode of *Knight Rider*, we discuss *American Idol* in real time on Twitter and Facebook.

So can we please stop acting like social networking and social media are inventions of the last 15 years or so? Sorry, I'll get off my soapbox now.

Share Your Thoughts

What are the trends and innovations that you think have done the most to change our social circles? If you had to pick one innovation or trend to be the single most important, which would you pick? Why? Share your thoughts in the comment section below. (Note: You must be logged in to comment.)

About the Author

Travis Robertson is a business strategist and Millennial consultant. He writes about entrepreneurship, leadership and millennials on his blog TravisRobertson.com. You can follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 17

The Convergence of Local, Social, and Mobile, Part 3: Mobile

Travis Robertson

It is only fitting that, as I sit here writing the third part of this series, I am doing it from a local coffee shop and bakery, tethered to my Android Evo 4G with Tweetdeck humming along in the background. The only thing that would be more fitting is if I were doing the next (and final) article in this series, where I will actually look at the convergence of local, social, and mobile, and how it will impact our lives moving forward (beyond just the regular trip to a coffee shop to work, I assure you).

For those who are joining us midway through this series, I would encourage you to start at the beginning. After all, for you to understand the implications of the convergence of local, social, and mobile, it helps to appreciate the somewhat siloed nature of those three areas, along with their evolution. That's what these first few articles examine.

We began with a historical look at the evolution of the concept of local, which you can read here. I wrote about how "local" was initially constrained by geography and travel limitations, but that has lessened primarily due to the rise of mass and personal transportation from the late 1800s through today. In the second piece, I wrote about how "social" evolved from being immediately local in nature to being an "always on" component of our lives, thanks to the evolution of communication technologies (again, primarily communication advances in the last 150 years or so).

This week, I want to look at the third component of this convergence and how it has evolved in its relatively short lifespan. It's time to place "mobile" under the microscope.

The Evolution of Mobile

Perhaps it's just me, but it feels peculiar to examine something so recent as mobile technology when we've spent the last two weeks on concepts as old as civilization itself. It almost feels like I'm writing an article in 1800 titled "A History of the United States and Its Impact on China." Anyone with even a small amount of historical knowledge would know that the United States as we know it was only about 25 years old. China? It was about 3,800 years old. Anybody attempting to write such an article would have been laughed out of business.

However, since I'm quite comfortable being mocked and I learned a long time ago not to take myself too seriously, I'm going to have a go at it.

Phase 1: Can you hear me now?

I sometimes wonder if Verizon stole that line from the very guy who invented the mobile phone as we know it today. I have to imagine he asked that repeatedly while testing out his invention. In my mind, I see the inventor, Martin Cooper (a researcher at Motorola), driving his colleagues crazy on a daily basis during development. (By the way, if you're younger than 35, click the link to check out a spectacular photo of Martin with an early-model cell phone. Awesome, I know.) If I were him, I would have been calling all of my friends in my office while testing. They'd pick up their phones with a resigned frustration, and I'd have a bit of fun at their expense.

The telephone rings. Jerry in Accounting rolls his eyes and sighs out loud. He's knee-deep in year-end reports and doesn't want another interruption. But he knows he can't ignore this call. Of course, they didn't have caller ID then, but he's been getting calls all day and he knows who's on the other end.

He picks up the phone and sighs into the receiver, "Hello. This is Jerry."

"Hey, Jerry! It's Marty! Guess where I'm calling you from!"

Jerry's pissed. "Where, Marty? Your workshop?" Then Jerry hears the sound of a toilet flushing and Martin sniggering in the background before he hangs up.

Jerry cries.

While that probably didn't happen, it's fun to think about. It's what I would have done. Which is probably why I haven't invented anything yet. Just guessing.

I was surprised to learn in my research that the very first phone call from a car occurred in St. Louis, Missouri, on June 17, 1946 (according to Wikipedia), prior to when Martin was making prank calls to his employees. However,

weighing in at 80 pounds and costing the equivalent of about $350 per month to operate, the equipment used for this first mobile call was impractical for widespread use.

But at that time, it was more about the possibilities than the applications. You see, prior to that first mobile call, it had only been possible to call people land line to land line. Radiophones were not allowed to hook into the public land lines. In fact, the ability to talk on land lines was in and of itself a huge leap forward in communications. As I discussed in my previous article on social, it gave people the capability to expand their social networks beyond geographic bounds—so much so that geography was becoming less and less relevant to the concept of "social."

And, while I can't promise you that Martin Cooper played practical jokes on his office staff, Wikipedia confirms that his first call on the handheld device was to his competitor at Bell Labs, Dr. Joel S. Engle, on April 3, 1973. So at least I know he did have a sense of humor (or at minimum, a wicked competitive streak).

Phase 2: The "G" wars

If the movie *Star Wars* were about mobile carriers, this is the point where the story would have begun. In 1979, the first 1G network launched in Japan, expanding to a number of countries before it launched in the United States in 1983. Companies were racing to get their networks up and running. Various standards emerged over the next 20 years, and devices became more and more powerful.

Mobile grew at astounding rates. According to Wikipedia: "In the twenty years from 1990 to 2010, worldwide mobile phone subscriptions grew from 12.4 million to over 4.6 billion, penetrating the developing economies and reaching the bottom of the economic pyramid." It's estimated that the penetration of mobile is above 97% in developed countries and above 50% in developing countries. In other X.com posts, we've looked at how developing nations are using their mobile devices for secure banking and transactions using services such as M-PESA and text messaging.

Companies are spending billions of dollars to expand their networks and improve their technology. The focus has shifted from the mobile device being simply a phone to the mobile device becoming a tool to interact with the world. Let me show you.

Phase 3: The phone gets a brain and learns to surf

If you want to understand how the smartphone is changing our lives and how we interact with the world, I would encourage you to spend time reviewing the following presentation from Matt Murphy and Mary Meeker. My friend Matthew Russell sent this to me and even wrote a quick overview of it here on the O'Reilly blog on X.com.

Top 10 Mobile Internet Trends (Feb 2011)

See the video of these trends at *http://www.slideshare.net/kleinerperkins/kpcb-top-10-mobile-trends-feb-2011*, as well as other presentations from Kleiner Perkins Caufield & Byers.

I won't cover everything in the presentation since you can flip through it yourself. However, there are a few points I want to highlight, as they're pertinent to this series.

Observation #1: Smartphones + Tablets > PC shipments

Please don't miss the significance of this equation. If you look at slide 7 of the presentation, you'll notice that in 2011, it's estimated that the shipment of mobile devices (both smartphones and tablets such as the iPad or the Xoom) is expected to outpace the shipment of desktop PCs and laptops. While older adults may still prefer surfing the Web with a larger screen or typing on a physical keyboard, more and more consumers are becoming comfortable with smaller devices that let them connect from anywhere.

The next major wave is mobile. This is no longer up for debate. The question is, *what are you going to do to capitalize on it?*

Observation #2: Social is driving mobile adoption

This is what we've been examining in this series. I talked about it in my previous article. Social has existed long before Facebook and Twitter. Socializing without restrictions and limitations is a basic human desire, and it's driving the adoption of mobile platforms. Take a look at slides 11–13 for more data on this.

On slide 13, notice the types of apps pushing the social interaction (other than Facebook). It boils down to three main categories: Information Sharing, Gaming, and Commerce. We spend a lot of time on this site discussing the trends of gaming and commerce on mobile devices and how PayPal is focusing on them for continued growth. But if you look at these three areas as part of a larger whole, you'll see a trend emerging: social commerce.

We'll discuss social commerce in the last article in this series when we begin to focus on the convergence of local, social, and mobile. But I want to stress something here: search isn't dead, but I don't believe it's the future. And by "search," I mean what we usually associate with SEO. I believe the future of commerce lies in social search and social commerce tied into our social networks. If I need a recommendation for a product or service, I first hit Twitter and Facebook. Then, if I need to do additional research, I search on Google. This used to be reversed.

In the past, social recommendations were hard to come by outside of your physically constrained (pre- Facebook and Twitter) networks. Sure, you'd ask a few friends who might know something. Now you can ask anyone on the Internet. Social commerce, social search, and social recommendation are going to explode. I think this is why you see Google putting local results above SEO results. Local is inherently more relevant than keyword optimization except in the research phase.

Observation #3: Time spent talking on smartphones is minimal

In fact, it's only about 32% of what the device is actually used for. The rest of the time is spent on social networking, emailing, web surfing, gaming, etc. The smartphone is less about the phone and more about the smart. Companies and developers need to notice this trend and start looking at how to empower users to engage with their services and products on mobile devices.

Observation #4: Mobile is revolutionizing commerce online and IRL

Most of us are already aware of how mobile commerce is growing online sales. More people are buying through their smartphones and other mobile devices. This is important, but we shouldn't get caught in the trap of thinking that mobile is a brick-and-mortar killer. In fact, look at slides 32–36.

With the rise in location-based services, shopping apps like ShopSavvy and RedLaser (owned by PayPal parent eBay), and promotion apps like Groupon, mobile has the capacity to increase sales at brick-and-mortar stores if utilized correctly. The key is to determine how to educate the average shop owner of the potential.

For those thinking of potential startup opportunities, re-read my last sentence. You're welcome.

Stay Tuned

We've looked at the three areas comprising the convergence—local, social, and mobile—so next we'll start delving into the convergence itself and how it will affect our lives moving forward. In the meantime...

Share Your Thoughts

Now that we've examined the three major components, tell us how you think their convergence will impact our lives in the next couple of years. (Note: You must be logged in to comment.)

About the Author

Travis Robertson is a business strategist and generational consultant. He writes about entrepreneurship, leadership, and millennials on his blog TravisRobertson.com. You can follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 18

The Convergence of Local, Social, and Mobile, Part 4: The Convergence

Travis Robertson

This is the fourth part in a four-part series. You can read Part 1 here, Part 2 here, and Part 3 here.

In September 2010, Wired Magazine ran a cover story with the bombastic title The Web Is Dead. Long Live the Internet. Needless to say, the piece was met with raucous debate about the validity of the viewpoint. If you read the article, you'll remember that the numbers are tough to argue with. The way the Internet is utilized has shifted from web browser traffic to distributed applications that run on devices such as the iPhone, iPad, Android, and even your desktop (Tweetdeck, anyone?).

For web developers (of which I was one for 12 years), the pronouncement of the death of the Web is understandably troubling. I also think it's a great way to sell magazines, which was unquestionably the intent of *Wired*'s editors. I'm not willing to go so far as to say that the browser is an antiquated piece of technology, waiting to wither up and pass on into obscurity, and I believe HTML5 could provide a much-needed boost to the admittedly clunky user experience the browser provides.

Furthermore, we still spend a tremendous amount of time with desktops or laptops, either at the office or at home. Corporations around the world are not likely to toss an iPad and iPhone at you and tell you to do your job—at least not anytime soon. And desktop apps are much more complicated to build since the expectations surrounding them are much higher; we have drastically different ideas about how functional a desktop app should be versus a mobile

app. So it's inherently easier for developers to shift focus to web-based development since it cuts down on overall costs as well as the frustrations of multiplatform development.

But to claim that the future of the Web or the Internet is tied to the browser experience is wishful thinking. In fact, most Internet traffic is already on mobile. There has been a rapid transition toward mobile, and every developer needs to understand the implications of this shift. It's happened in part because the browser is an inherently bad method for data consumption on mobile devices. Even on the iPhone and Android smartphones, the browser is not nearly as useful as native apps for consuming data.

This is why we need to understand the convergence of local, social, and mobile, and the impact it will have in the very near future. This is the fourth and final part in a series I've been doing called The Convergence of Local, Social, and Mobile. In the previous articles, I evaluated the somewhat siloed nature of these three components and examined their evolution. Now it's time to put their convergence under the microscope.

First, A Look at the Data

I have the smartest readers on the Web, so I figure a little geeking out over some stats won't bore you. (If I'm wrong, feel free to skip ahead. I won't be offended.) Here are five stats I think are relevant to our discussion:

- **Browser traffic accounted for only 23% of total Internet traffic in 2010.** The majority of traffic was from video (51%) and peer-to-peer (23%), for a combined total of 74% of all Internet traffic. According to the *Wired* article, "The applications that account for more of the Internet's traffic include peer-to-peer file transfers, email, company VPNs, the machine-to-machine communications of APIs, Skype calls, World of Warcraft and other online games, Xbox Live, iTunes, voice-over-IP phones, iChat, and Netflix movie streaming. Many of the newer Net applications are closed, often proprietary, networks."
- **Shipments of smartphones and tablets combined will outpace desktop and notebook shipments combined in 2011.** You can see this statistic illustrated in slide 7 of this presentation from Matt Murphy and Mary Meeker on the top 10 mobile Internet trends. Needless to say, this represents a major shift in how people consume data. And, as I mentioned previously, the Web in its current form is not suited for mobile devices like smartphones. It could be argued that tablets bridge the gap. However, native apps are still much better than the browser, but I'm hopeful that HTML5 will allow for changes here.

- **Social companies such as Facebook and Twitter have strong mobile usage.** In fact, Facebook boasts about 200 million active mobile users and reports that those users are twice as active as the desktop-only crowd. Twitter reports that over 40% of all tweets come from mobile users. Mobile represents the way that Millennials (people 34 and under) interact with the world.
- **Online ecommerce represents about 5% of total retail sales in the US; mobile ecommerce is expected to reach that number much faster.** In fact, mobile ecommerce is at about 1% right now, and it's really only been around since the end of 2006. It's estimated to rise to 2% of all sales by the end of 2012. There is enormous opportunity in mobile ecommerce, which we'll cover in more depth shortly.
- **Mobile shopping apps alter consumer behavior and drive traffic to offline locations.** Stop and think about that for a second. Online commerce (ecommerce) has been ruthless to brick-and-mortar merchants. As I'm writing this, we're only a couple of days past Borders bookstores filing for bankruptcy. Speculation has been swirling around the future of physical stores for a long time. Now, apps like Shopkick are finding ways to revitalize brick and mortar. In the aforementioned presentation, you can see this trend illustrated on slide 35. The stunning part of the slide is that the increase in foot traffic occurred on Cyber Monday and was even higher than Black Friday's foot traffic.

What Does This Mean for the Future of Local, Social, and Mobile?

Well, in all honesty, it means a lot—especially where they converge. However, I want to focus on the five key areas that I think are most critical to understand:

- **Ecommerce will be decentralized to the edges.** In the early days of ecommerce, the focus was on the website; it was the center of ecommerce. And, while it will continue that way for some time, we're going to see mobile rise very rapidly in this space and (I believe) surpass browser-based ecommerce—likely in the next five years.
- **Heavy innovations in the speed and ease of mobile commerce will allow for its rapid growth.** The biggest challenge for mobile ecommerce has been the practicality of payment on handheld devices. Stepping through the checkout process on a mobile device is about as pleasant as being kicked in the face with metal cleats. However, with tools like PayPal's Mobile Express Checkout, the pain and friction is drastically reduced. By greasing the skids of mobile commerce, PayPal makes it possible

for us to pay for products on the devices we never leave home without: our smartphones.

- **We will witness a revitalization of brick and mortar.** All of this is pointing toward a revitalization of the brick-and-mortar shopping experience. No doubt, the Web is highly convenient. Even more convenient is the mobile experience, which is always with us. However, as part of the decentralization process, savvy retailers will not look to undercut their physical locations, as they are heavily invested in them. Instead, they will use check-in services, deal services, coupon services, and social tools to drive foot traffic. This, by the way, doesn't mean that all brick-and-mortar stores will be able to adapt. Far from it. But the opportunity is available —for those who seize it—to snatch sales away from the online marketplace.
- **We will finally see shifts toward more normalized pricing across online and offline locations.** Maybe this is a little bit of wishful thinking on my part, but I believe this is the only way for companies to protect their heavy investments in physical locations. By offering such drastically reduced online prices, companies are effectively undercutting and harming themselves. The book industry is a spectacular example of this. Last week, I needed to buy a book for a friend. I would have preferred to pick it up at the local Barnes and Noble and hand it to my friend when we met that afternoon. I had a $20 Groupon and wanted to use it to buy the book. I knew I could get the book online for $13–14. When I called the store, they had the book in stock, but it was priced at $26. I decided I didn't need the book that badly and ordered it. This is a terrible business model: margins are lousy and stores are closing. Apps like Shopkick, Groupon, and Foursquare, which help close the gap between offline and online pricing and have the added benefit of no delivery wait time, will continue to drive buyers toward local, in-store purchases.
- **Watch for the integration of mobile commerce with our social networks.** Our social networks provide immense value to us during the research phase of purchasing goods and services. Many Millennials begin their research phase on Facebook and Twitter. Having more of that data immediately available while conducting commerce locally would provide unparalleled value and huge opportunity to those who manage to integrate that data. Imagine if I could scan the barcode of an item I was interested in and immediately read reviews by those in my social circle on my smartphone. Then, I could quickly comparison shop on the phone and place an order using Mobile Express Checkout from PayPal. Mark my words: it's coming very soon.

Share Your Thoughts

There is so much potential when we talk about the convergence of local, social, and mobile. The question isn't whether it will change our lives, but how. So I open this up to you: how do you see the convergence impacting our lives in the near future? Share your thoughts in the article's comments section on x.com.

About the Author

Travis Robertson is a business strategist and generational consultant. He writes about entrepreneurship, leadership, and millennials on his blog TravisRobertson.com. You can follow him on Twitter (@travisro), where he spends far too much time.

CHAPTER 19

To Catch a (Quantum) Thief

Dan Watkins

Data Security in the Quantum Age

When you drill down on the "spooky" world of quantum computing, you are confronted with the absurd merry-go-round that is life—you don't quite understand what's going on, but it's fun.

The concept itself upsets the calculator that is generally believed to underscore reality. However, there appears to be a deeper, more fundamental, and decidely weirder layer to existence. In the atomic or subatomic realm, particles have been observed behaving in a way that conflicts with the physical laws as they have been traditionally defined, thus Einstein's famous description: "spooky action at a distance."

From a computing standpoint, if a bit of data is either a "1" or a "0" in a classic computer, that same bit can be simultaneously a "1" and a "0" in the theoretical quantum computer. When very smart people build upon that revolutionary insight, they are naturally excited about the potential commercial applications of quantum computing.

The most obvious application is in massively expanding the data-crunching capabilities of the worldwide digital ecosystem. As the data processing boundaries for the Internet, cell phones, and payment systems continue to be pushed back, the demand for more processing power and speed only grows.

Quantum computing has been touted as a solution to this demand. It might be a tad hyperbolic when scientists claim it would take a classic computer the size of the universe to perform as many computations as a single quantum computer, but it's the thought that counts.

The Downside

Although all great technological breakthroughs have the potential for changing the world for the better, the opposite is also true. Like the two-sided coin of nuclear power, quantum computing as a solution for our data processing needs has a rather large potential downside—in data security.

"Quantum computing would massively increase the compute power scientists and engineers have at their disposal," says Karsten Nohl, white hat and graduate student at the University of Virginia. "Like most tools, these same capabilities can be used for criminal activity. In particular, most cryptographic problems that are uncrackable today would become solveable eventually with quantum computers."

Cryptography is at the heart of how data is protected as it is transported over electronic payment systems. When a consumer swipes his debit card through a reader at a convenience store, for example, the data on that card, such as the cardholder's account number and PIN code, is encrypted with a cryptographic algorithm that renders that data meaningless if a fraudster should steal it. To read the actual account information, the fraudster would need the algorithmic key to unlock the code.

Barring that, the fraudster could subject the encrypted information to a program that analyzes all the possible permutations of the code to eventually crack it. Depending on the sophistication of the encryption, cracking that code could take millenia, according to Hadi Nahari, Principal Devices & Security Architect at PayPal.

The current standard for data encryption is Triple DES. Nahari describes it as using one key to encrypt data, a second key to decrypt it, and a third key to encrypt it again. "So the idea is every time you add this level of complexity, in this case with another layer of encryption, another layer of indirection, you add to the extent of computation," he says.

However, if the fraudster had access to a quantum computer, Nahari says that encryption scheme may not be as secure. Current encryption schemes "rely on the assumption that the amount of computation required to solve this problem is so big that with the current computers, it would take, say, two or three thousand years if all computers on the face of the planet were dedicated to solving this problem," he says.

Not so with a quantum computer, where massive processing power brought to bear on the encrypted data would reduce the computation time to a comparably tiny and manageable fraction, from several hours to perhaps a day.

Bad News

It was recently revealed that RSA's SecurID two-factor authentication scheme was hacked. SecurID, a popular technology used widely across the Internet, enhances usernames and passwords (factor one) with tokenization technology (factor two) to make it more secure when users log in to programs or networks. In RSA's case, the SecurID token is a randomly generated authentication code that users type in when logging on.

The breach apparently didn't result in the theft of user data, but the hackers may have obtained a "blueprint" to how SecurID works—knowledge that may be exploited in the future. Before the breach was announced, Nohl said the RSA algorithm would not hold up under a quantum computer onslaught.

Nahari generally agrees with that position. "RSA algorithms are based on the power of prime numbers and the factorization of prime numbers," he says. "So if you have very large prime numbers, and a set of two large prime numbers, if you multiply or do some mathematical calculation with these two prime numbers, if you knew what these prime numbers were, it's going to be very easy to factor and come up with the components of that very large number.

"For someone who doesn't have that knowledge, they would have to go through a lot of mathematical calculations to be able to factor very large numbers. But for this type of calculation, then quantum computing would be very bad news for the RSA algorithm."

Hacking Accelerated

The sophisticated RSA hack seems to speak to the future trend in electronic fraud, and how quantum computing would accelerate it. The RSA hackers apparently weren't interested in hacking passwords to get to accounts; instead, they went after the encryption architecture itself, because it is in the back-end systems where the data stream is deepest—and potentially richest.

Nahari says, "If I have that [quantum computing] device and I know the way that people connect to the back end to all the juice, all the data, all the data centers, all of those massive amounts of user data that's stored... [I] mount a man-in-the-middle attack and just get some data about this channel and then break into the passcode, break into the encryption algorithm and get the key, the private key, or the symmetric key of that channel, and then if I could use that to impersonate one of the admins of the system, then I can escalate my privileges and gain access to the back end of the system.

"It would be a better way to drill into the back end and get the real data and not really worry about user names and passcodes of the users who own this data... If I can get that encrypted data and I know that I could decrypt it with my quantum computer in a matter of hours and sell this information, then perhaps I will do that instead of obtaining access to user names and passwords."

Game Changer

It is clear that a fraud ring or a rogue nation with access to a quantum computer poses a serious threat to data security. However, Nahari is quick to point out that, if and when quantum computers arrive, they will be available to the good guys as well.

"Yes, if a quantum computer only goes into the hands of the hackers, then it's going to be really nasty, really fast," he says. "But it's going to be accessible to both the police, the cops, as well as the criminals. It doesn't mean the game is going to change."

In short, the game won't change because both sides will have access to the same technology. The bad guys will have massive computational power at their disposal to crack encryption schemes. But the good guys will have that same capability to devise ever more elaborate and computationally massive encryption schemes to counter attacks.

Nahari believes the era of quantum computing is nearing, out of necessity. The energy requirements necessary for processing and storing data worldwide grows more burdensome by the day.

"Take a look at the terabytes of data that Google crunches every day," he says. "How many years do they want to build their own Linux-based servers, annexes here and there? At some point it's not going to be scalable. Google spends a humongous amount of money on their power bill for their data centers. At some point physics and the laws of gravity govern that we will have to have more scalable and more optimized computing paradigms."

5 to 10

It seems that predictions for when quantum computing will arrive are always either 5 or 10 years away. Nohl said the last time he studied quantum computing was 5 years ago, when it was predicted the new technology was about 5 years away. While it hasn't arrived yet, it may happen sooner than later.

In October 2010, Dr. Geordie Rose, founder and CTO of the Burnaby, B.C., startup D-Wave Systems, presented his company's findings at a Google Tech Talk. D-Wave has actually built a quantum computer based on what Rose calls "quantum annealing" technology.

The computer looks like a NASA deep space probe, with a series of circular "plates" stacked vertically on a cylindrical core. The quantum processing chip —very much like a standard superconducting chip and made with an exotic material known as niobium, which is highly sensitive to temperature and magnetic fields—is mounted on a kind of motherboard on the bottom of the device and kept at a cryogenic temperature of 20 milliKelvin, or 100 times colder than interstellar space. The whole mechanism, shielded from the elements in a metal cylinder inside a refrigeration unit called a Pulse Tube Dilution Refrigerator, takes up about 200 square feet and is accessed remotely by D-Wave researchers.

If such a device allowed a corporation to fit all of its data processing capability into one room, Nahari is certain PayPal would be interested. In fact, a little more than interested. "If it becomes semi-commercially available, PayPal would be jumping up and down," he says.

CHAPTER 20

10 Best Practices for Employing QR Codes

Brenna Roth

In the last several years, the proliferation of smartphones has led to a breakthrough in online marketing. To better communicate with their customers in this new digital marketplace, many cutting-edge entrepreneurs and marketing firms are turning to Quick Response (QR) Codes to convey information with greater speed, efficiency, and convenience than ever before.

QR Codes are a known commodity with unknown potential. They were developed in Japan during the mid-'90s, and they are an integral part of that country's high-tech cellphone culture. More recently, savvy artists, guerilla marketers, and business owners have adapted QR technology for use in the American and European markets.

So what is a QR Code? On the most basic level, a QR Code is an arrangement of black modules on a white background within a square measuring at least 1.25″ by 1.25″. It is a matrix barcode, meaning it's two-dimensional, and the arrangement of modules is unique to each code. Once scanned, these modules transmit data to the scanning device—either a smartphone or dedicated QR scanner. The potential for this data is limited only by the imagination of the code's creator; usually, it links to a website where the customer can purchase or learn about a product, service, or event.

One of the primary advantages of QR Codes is access to advanced metrics. This access ensures that you know who is visiting your website and how they found it—vital information to consider as you formulate your marketing strategy. Furthermore, many QR Code generators can be utilized free of charge, lowering overhead costs and maximizing your profits.

The following list outlines 10 of the best practices for employing QR Codes in your business or marketing campaign. Follow these 10 items, and you will be well on your way to creating an innovative, effective, and efficient marketing strategy to grow your business.

Size Matters (the Bigger, the Better)

Make sure the QR Code is big enough to scan easily. The minimum size should be 1.25″ by 1.25″; any smaller than that, and the code becomes difficult to scan. QR Codes were created to make it easy to link to additional content, so don't lose people because they can't scan your code. If the QR Code is not large enough, it may not scan properly and valuable customers may never see your content. A small, difficult-to-scan QR Code frustrates everyone, businesses and customers alike. Don't make that mistake. If you make absolutely sure your QR Code is big enough to read easily, you will reap the rewards of additional page views and downloads. When in doubt, scan the code with several different phones to make sure that it will work.

Short Is Sweet

QR Codes are an efficient and artistic-looking way of storing data. Try to store the least amount of data possible by using short URLs. This makes a code easier to scan, and a less cluttered code is more visually appealing. There are a number of good (and free) URL shorteners online, such as bit.ly, goo.gl, and tinyurl.com. For added features, try memurl.com, which creates easy-to-remember mnemonic links, or dwarfurl.com, which uses advanced statistics to track the traffic to your site. Use these services to clean up your codes and make things easier for the people who want to scan them.

Location Matters

Make sure to put your QR Codes in a place people can use them. This means thinking through where people will be scanning your codes and making sure the code is easy to access. Excellent placement options include newspaper or magazine pages, flyers or other promotional literature, and easy-to-spot locations on your website. Putting a QR Code online is a good move to direct traffic to your website, but make sure the customer can easily find the code—place it on the top of the page or fixed in the margin, for example. Don't make people jump through hoops to scan your code. Make it accessible and easy to use.

Know Why You're Using It

What are you trying to accomplish with your QR Code? Are you looking to provide additional information to users? Are you providing the code as an easy way to download something? For a QR Code to be utilized effectively, you need to know why you're using it. Place it on a print advertisement to direct customers to your online store. Place it on a flyer for your band to make it easy for your fans to purchase concert tickets or merchandise and download your music. Place it on an invitation to a wedding or baby shower, and your guests will be able to easily access the directions and gift registry. Once you know why you're using it, make sure that everything else fits that usage strategy.

Make It Count

Don't use your QR Code to repeat things you've already said. Use the code to get to the next step in the customer engagement cycle. Provide meaningful content that reflects the marketing strategy you decided upon in point 4. Successful marketing is goal oriented and measured with quantifiable metrics. Once you know what your goal is for the QR code, you need to make sure that the content backs that up and helps accomplish that goal. At the same time, you need to make sure you have a system in place to track how the QR Code is affecting traffic to your site. That way, you'll know if you're placing the code in a suitable location or if the code should be moved.

A Little Goes a Long Way

Don't throw a million codes on things (unless there is a compelling reason to do so). Make sure you maximize your codes' effectiveness by using them only when needed. One QR Code can help bring a document to life and give you a chance to further engage with people who scan the code. Overdoing it can frustrate users and help ensure that the important codes don't get scanned because people scanned a different code and didn't find what they were looking for. One benefit of QR Codes is that they take up relatively little space when compared to a traditional print advertisement. Use metrics to ensure efficiency; you're shooting for the highest amount of page views for the lowest amount of print space.

Tell People How to Use It

QR Codes are far from ubiquitous. Give people directions on how to use them. The directions don't need to be complicated; "Scan this with a QR Code reader" should suffice. Providing information on how to download the appropriate scanning app helps take things a step further and makes sure you're not losing people who might benefit from scanning your code.

Provide an Alternative

Don't assume that everyone has the ability to scan QR Codes. Not everyone has a smartphone, and some people who do may have broken cameras, uncharged phones, no service, or other technical difficulties. Provide a short URL that people can type in manually (or easily remember) if they are unable or unwilling to scan your code. The idea is to make things as easy to use for as many people as possible. Offering a simple fallback method makes sure you're maximizing who sees the additional content to which you link your QR Code.

Respect the Border

A QR Code comes with a white border around it. Respect that border. The border is an integral part of the code; the code will not scan properly if the border is infringed upon. In addition, the border is what sets your QR Code apart from the content on the page around it, and the code may not be visible without it. Be careful not to overlap that white space with other images or text.

Bring the Wow

The most important thing to do is to provide an awesome experience. Incorporate images, audio, video, and interactivity into the website to which your QR Code links, as these components can't all be duplicated on paper. This experience needs to reflect the marketing and usage strategy that you developed in point 4. Make sure you're engaging the person scanning your code in a way that provides a stimulating visual and audio experience. If you don't, you can bet your competitors will.

About the Author

Brenna Roth is the cofounder of Think Roth, a mobile marketing company that gives professionals the tools they need to run engaging QR Code campaigns.

CHAPTER 21

Integrating Payments into WordPress, Part 1: Features and Getting Started

Bill Day

I've explored a number of ways to integrate the power of the PayPal X Platform into your applications in my DevZone articles over the last few months. Recent installments have shown you how to accelerate your Adaptive Payments development using the Apigee PayPal API Console and how to buy and sell digital goods in data markets using PayPal Embedded Payments and micropayments.

Now I'd like to explore how to use PayPal payment solutions in the world's most popular blog and content management system (CMS) software: WordPress.

In this article, I'll introduce WordPress (commonly abbreviated as "WP" by WordPress developers and power users, aka "Pressers") and then take you on a whirlwind tour through installing and using it. Future articles will introduce the WordPress plugin model, through which you can greatly extend the capabilities of your WP-powered system. They will also discuss PayPal-based WP plugins and how to write your own payments plugins.

What Is WordPress?

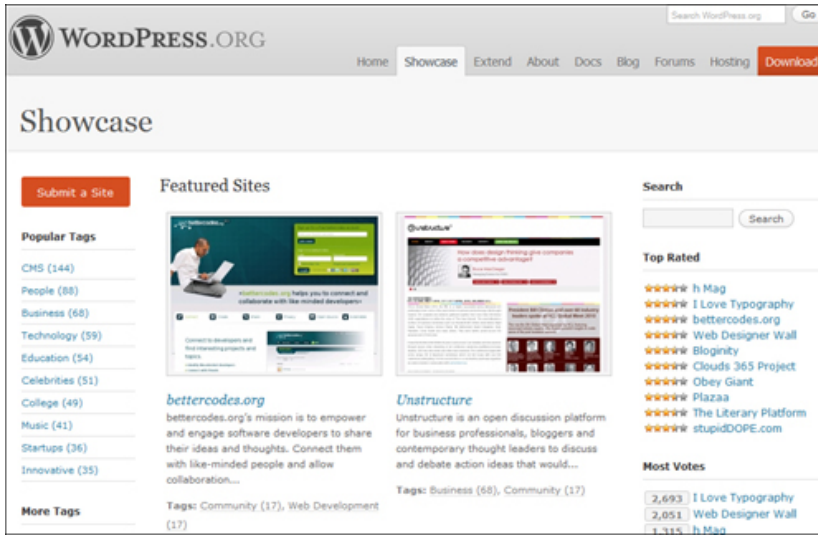
WordPress.org, the central WP development and Presser community hub, describes WordPress thusly:

> *WordPress is web software you can use to create a beautiful website or blog.*

In essence, WordPress is a freely available, open source, PHP- and MySQL-based web publishing system. It's very customizable out of the box, and can be further extended as needed through a little PHP hackery. The WP home page goes on to note:

> *The core software is built by hundreds of community volunteers, and when you're ready for more there are thousands of plugins and themes available to transform your site into almost anything you can imagine. Over 25 million people have chosen WordPress to power the place on the web they call "home."*

You can get a feel for what's possible by visiting the WordPress Showcase page and looking through a few of the thousands of sites powered by the software. Click here to check out the Showcase, also shown below. Go ahead, I'll wait.



Pretty impressive, right? Now that you have a feel for what's possible, let's examine how to implement a site using WP.

WordPress Documentation

First you should visit the WordPress Codex, the source of official documentation for the software, its installation and usage, and the development of WP add-ons such as plugins and themes. The Codex contains the following sections:

- Getting Started with WordPress: Describes where to start, how to install the software, how to publish your first posts, and how to manage files and plugins; offers a "New to WordPress" FAQ; and provides a section on the latest release family.
- Working with WordPress: Contains a lot of useful information on WP features and how to use them, theme and plugin administration, creating backups, fighting comment spam, and much more. Plan to spend some time here if you're serious about getting the most out of WP.
- Site Design and Layout: Dives into blog design and layout issues with an emphasis on CSS, themes, and templates (all used extensively by WP).
- Advanced Topics: Offers information on multiuser WP, deeper details on important WP concepts such as "The Loop" that displays each post, and tips for optimizing your installation.
- Troubleshooting: Provides FAQs galore and explains how to find answers when you're having problems.
- Developer Documentation: Includes database details and a discussion of the plugin API, WP cookies and coding standards, and how to use PHP to bend WP to your will. We'll be discussing this information in more detail throughout this series.
- About WordPress: Contains WP contributor acknowledgments, version information, WP community terminology, information on how to contribute to WP, documentation and Codex information, and more.

There are a couple of pages linked to from the aforementioned sections that I want to specifically call to your attention now, before we dive into installation, usage, and plugin extensions.

The first is the list of WP features. If you have a question about whether or not WP can do something, this is a good place to start your search for the answer. The features page also lists the currently required prerequisite software including release numbers (as of this writing, PHP 4.3 or newer and MySQL 4.1.2 or newer, but these are shifting targets as new releases are made). Note that you'll also need a web server that supports PHP and MySQL (I prefer Apache and will use it for the rest of this series).

In addition to exploring the features and prerequisites, you might also want to spend a little bit of time at this point reading up on WP terminology and jargon. Grounding yourself in the language used by Pressers will help to accelerate your learning process and remove unnecessary barriers as you move along.

Installing WordPress

Enough talk, let's install! WP is famous for its easy installation. In fact, many web hosting companies have easy installation processes that will automatically install WP for you with minimal fuss. If your host offers this, I'd suggest trying it.

If your host doesn't provide a WordPress auto-installer, then you can follow the steps in the "Installing WordPress" Codex page. In brief, you need to:

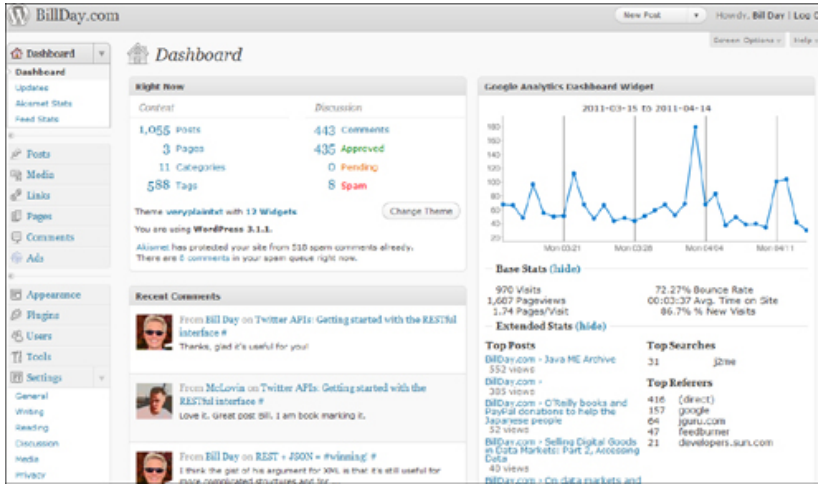
1. Download and unzip the WordPress package (click here for the latest stable release).
2. Create a MySQL user and database for use by your WP installation.
3. Rename the WP configuration file, `wp-config-sample.php`, to `wp-config.php` so you can begin editing it (you might even want to copy it to the new filename rather than renaming it; that way, you can reference the initial contents of the file after you make changes to the copied one).
4. Edit `wp-config.php` in a text editor to configure the database access information you just created in the second step.
5. Place the WP files where you want them on your web server (putting them in the root of your domain has its advantages, but there may be reasons for putting them in a subirectory of your site; whichever you choose, note the location now because you'll need it later).
6. Remember how I just told you that you needed to know where you put your WP files? You'll use that information here. You need to execute the WP installation script by accessing `wp-admin/install.php` in a web browser. If you installed WP in a subdirectory on the example.com domain, for instance, you could access the script at *http://example.com/subdirectory/wp-admin/install.php*; if you installed WP to the example.com domain root, the script would be at *http://example.com/wp-admin/install.php*.

Additional installation details are available from the "Installing WordPress" page.

First Steps After Installation

Once you've installed WP, you're ready to take it for a spin. The Codex provides a great tutorial for your first time, aptly named "First Steps with WordPress."

This tutorial will walk you through the basics of logging in to the WP administration interface and performing various admin actions. This includes a look at the WP dashboard, which summarizes key information when you first log in. For instance, here's my own site's dashboard (note that I use several plugins such as the Google Analytics plugin, which displays key metrics in my dashboard as pictured here; I'll discuss these plugins in the next article in this series).



The tutorial goes on to illustrate how to change the look of your content using themes (click here to dive deeper into themes and here for more on WP's use of CSS for styling). It also details how to write and publish posts, how to manage comments that your site visitors (readers) make on those posts, and much more.

I strongly encourage you to spend some time working your way through "First Steps with WordPress."

Coming Next Time

In this article, I've introduced the WordPress system. I've shown you how to install and get started using WP. I've also pointed you to a number of references to learn about WP and how to best take advantage of its capabilities.

In my next article, I'll introduce you to the power of WordPress plugins. I'll look at several PayPal-based WP payments plugins (think shopping carts) and discuss pros and cons of each. I'll also identify payments functionality that might be missing from the most popular plugins, and I'll use that information in a follow-up post discussing how you can use PHP to create your own PayPal X Platform–based WP payments plugin.

In the meantime, I'd greatly appreciate your questions, suggestions, and requests for this series. Please leave a comment below to let me know what you'd like to see covered in future articles.

About the Author

Bill is founder of Day Web Development, where he provides technical writing and editing, specification development, and platform and technology evangelism services. Recent projects include writing source code–level documentation, articles, partner how-to materials, conference session descriptions, and blog entries for O'Reilly Media, NVIDIA, Nokia, and Digital Reasoning Systems. In addition to being an active writer and blogger for the PayPal X Developer Community, Bill is also a contributor to GeekDad.com. Learn more and contact Bill via his LinkedIn profile and BillDay.com.

CHAPTER 22

Integrating Payments into WordPress, Part 2: PayPal Plugins

Bill Day

In my previous article, I introduced WordPress (WP), provided links to documentation to help you get up to speed with it, and then showed you how to install and start using it.

This time I want to describe the WordPress plugin model. I'll make some recommendations for plugins that are not only useful, but also instructive as you learn about what's possible in WP. Then I'll examine some of the most popular and highest-rated PayPal plugins according to WP users. I'll also identify payments functionality that might be missing or lacking in currently available plugins; we'll go deeper into that functionality in a future article.

Extending WordPress with Plugins

As the WordPress Codex succinctly puts it:

> *Plugins are tools to extend the functionality of WordPress.*

That's a high-level definition for WP users, but what about a workable definition for programmers and hackers? Here's a better description from the Codex "Writing a Plugin" page:

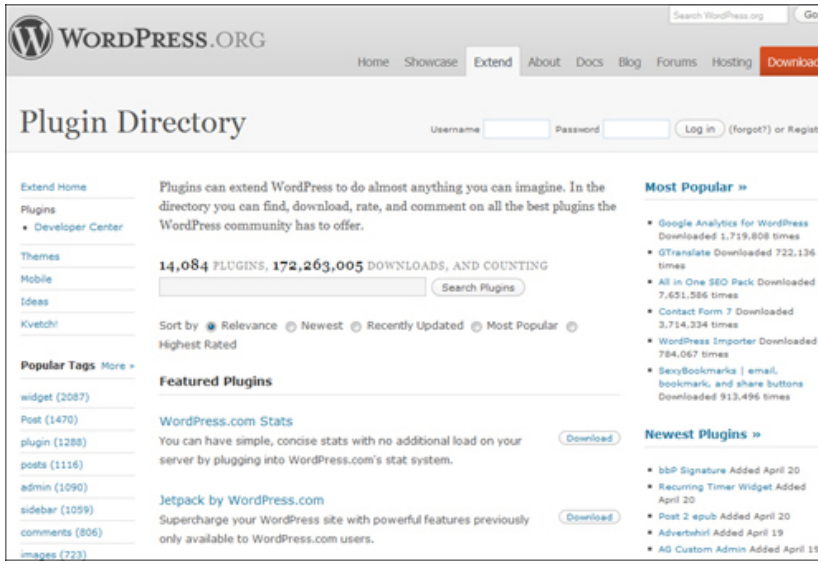
> *A WordPress Plugin is a program, or a set of one or more functions, written in the PHP scripting language, that adds a specific set of features or services to the WordPress weblog, which can be seamlessly integrated with the weblog using access points and methods provided by the WordPress Plugin Application Program Interface (API).*

The Plugin API page goes on to provide a lot of useful information on the hooks you can use to tie plugins into WordPress. These hooks falls into two basic categories, actions and filters. In a nutshell, *actions* are event-based hooks (WP launches the appropriate plugin when a specific action occurs), whereas *filters* are text-modification hooks (WP invokes these to modify text before writing it to the WP database or sending it to a client browser).

Understanding how to hook a new plugin into WordPress is key for any developer interested in extending the WP system with his own functionality. The preceding links provide enough information for you to explore this further on your own, and in the next installment of this series we'll dive deeper into plugin development together. Before we do that, however, let's look at some popular plugins, both general-purpose and PayPal-related options, so you have a good understanding of what's available and what those plugins can do.

Plugin Recommendations

The definitive place to search for existing plugins is the WordPress Plugin Directory, shown here.


WordPress.org
Home
Showcase
Extend
About
Docs
Blog
Forums
Hosting
Download
Plugin Directory
Extend Home
Plugins
Developer Center
Themes
Mobile
Ideas
Kvetch!
Popular Tags More »
widget (2087)
Post (1470)
plugin (1288)
posts (1116)
admin (1090)
sidebar (1059)
comments (806)
images (723)
Plugins can extend WordPress to do almost anything you can imagine. In the directory you can find, download, rate, and comment on all the best plugins the WordPress community has to offer.
14,084 PLUGINS, 172,263,005 DOWNLOADS, AND COUNTING
Search Plugins
Sort by Relevance Newest Recently Updated Most Popular Highest Rated
Featured Plugins
WordPress.com Stats
You can have simple, concise stats with no additional load on your server by plugging into WordPress.com's stat system.
Download
Jetpack by WordPress.com
Supercharge your WordPress site with powerful features previously only available to WordPress.com users.
Download
Most Popular »
Google Analytics for WordPress Downloaded 1,719,808 times
GTranslate Downloaded 722,136 times
All in One SEO Pack Downloaded 7,651,586 times
Contact Form 7 Downloaded 3,714,334 times
WordPress Importer Downloaded 784,067 times
SexyBookmarks | email, bookmark, and share buttons Downloaded 913,496 times
Newest Plugins »
bbP Signature Added April 20
Recurring Timer Widget Added April 20
Post 2 epub Added April 20
Advertwhirl Added April 19
AG Custom Admin Added April 19

As of this writing, the directory contains 14,084 plugins. The directory allows you to search for plugins to meet whatever need you have for extending WP. It allows you to filter plugin search results based upon relevance, how recently plugins were added to the directory, how recently they were updated in the directory, how many WP users have downloaded them (popularity), and how highly users have rated them.

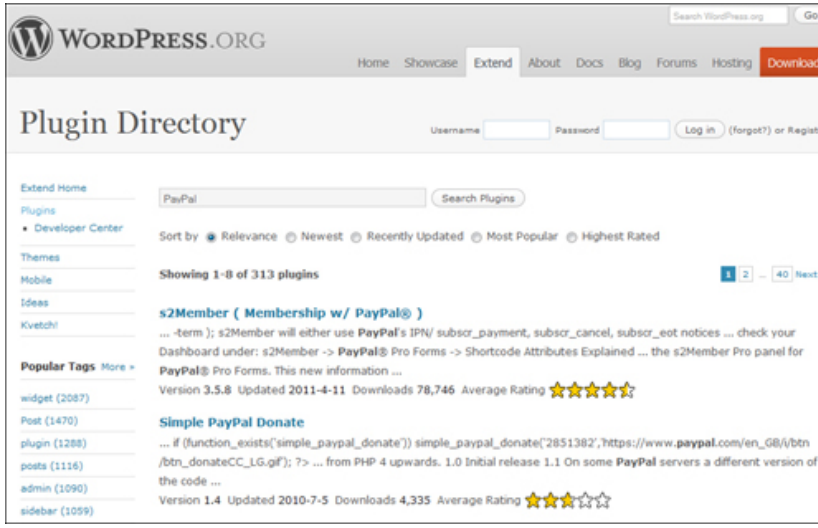
To give you a better feel for the sort of things that plugins might provide, I've listed a few of my favorite WP plugins below. I use these on BillDay.com and recommend them to others looking for similar functionality for their WP-powered sites. Click the plugin name to visit that plugin's page in the directory to learn more about it or install it yourself.

- Advertising Manager: Makes it easier to administer Google AdSense and other advertising and referral blocks.
- Akismet: Protects your blog from comment and trackback spam by connecting to the Akismet server and comparing with spam contributions from millions of other users (requires free Akistmet key to activate).
- FeedBurner FeedSmith: Redirects your WP feeds to a FeedBurner feed so you can learn about your subscribers. (A link is not currently available; I'm not sure if this has been updated under a different name or not.)
- Feed Stats for WordPress: Provides an easy way to view statistics from your FeedBurner feed in the WP Dashboard.
- Google Analytics Dashboard: Shows a summary of your Google Analytics–tracked activity in the WP Dashboard.
- Google Analytics for WordPress: Makes it easy for you to add Google Analytics tracking to your WP-powered site.
- PostRank: Enables you to quickly see which of your WP posts are most popular with readers, track social media analytics, and engage with site visitors from your WP Dashboard.
- ShareThis: Empowers your visitors to share a post or page by using email or posting to a great many social networks.
- Twitter Tools: Integrates your WordPress blog and Twitter, allowing you to import tweets into WP and/or send blog posts to Twitter; you can also optionally show your tweets in your WP sidebar and post tweets from the WP administrative interface.
- WordPress Hashcash: Includes JavaScript that blocks spam bots.

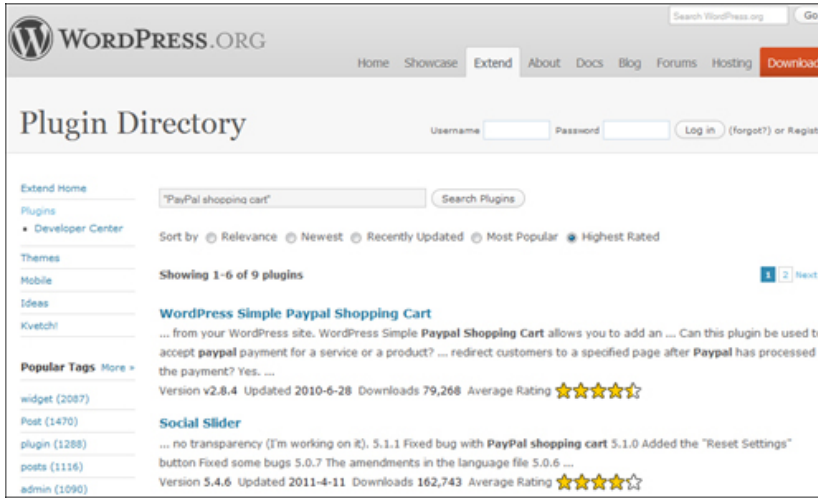
PayPal Plugins

Now that you know what plugins are and how to find them in the WP directory, let's search for PayPal-related plugins to see what's currently available.

Simply doing a naïve search for all possible related plugins using the term "PayPal" yields 313 plugins, as shown below, in the results. That's a few too many to discuss individually in this article!



Let's limit the search more by changing the search term to "PayPal shopping cart." That gives us a much more manageable nine plugins to consider. From there we can sort by popularity and user rating, as shown in the next image.


WordPress.org
Home Showcase Extend About Docs Blog Forums Hosting Download
Plugin Directory
Username Password Log in (forgot?) or Register
Extend Home
Plugins
Developer Center
Themes
Mobile
Ideas
Kvetch!
"PayPal shopping cart"
Search Plugins
Sort by Relevance Newest Recently Updated Most Popular Highest Rated
Showing 1-6 of 9 plugins
WordPress Simple Paypal Shopping Cart
... from your WordPress site. WordPress Simple Paypal Shopping Cart allows you to add an ... Can this plugin be used to accept paypal payment for a service or a product? ... redirect customers to a specified page after Paypal has processed the payment? Yes. ...
Version v2.8.4 Updated 2010-6-28 Downloads 79,268 Average Rating
Social Slider
... no transparency (I'm working on it). 5.1.1 Fixed bug with PayPal shopping cart 5.1.0 Added the "Reset Settings" button Fixed some bugs 5.0.7 The amendments in the language file 5.0.6 ...
Version 5.4.6 Updated 2011-4-11 Downloads 162,743 Average Rating
Popular Tags More »
widget (2087)
Post (1470)
plugin (1288)
posts (1116)
admin (1090)

From these searches, we see that "WordPress Simple Paypal Shopping Cart," shown in the following figure, is the highest-rated and the second most popular PayPal shopping cart plugin. You would click on the plugin title to examine its page and then download it if you're interested in installing it.

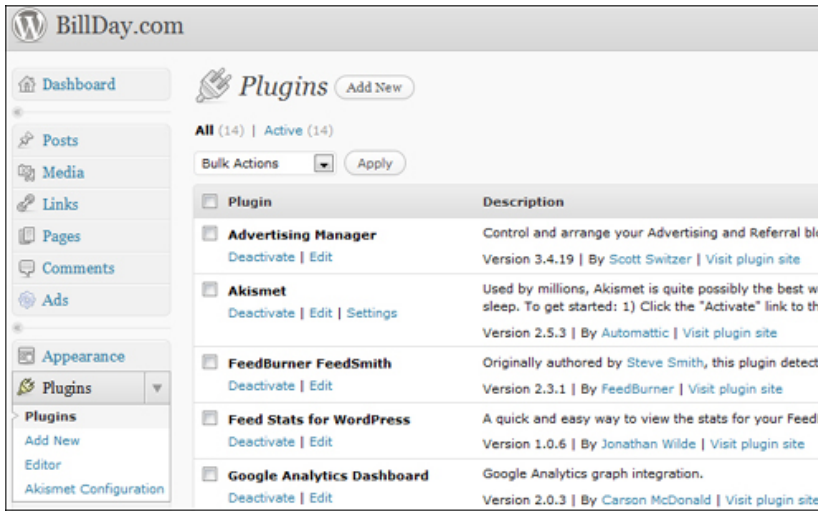
WordPress.org
Home Showcase Extend About Docs Blog Forums Hosting Download
Plugin Directory
Username Password Log in (forgot?) or Register
Extend Home
Plugins
Developer Center
Themes
Mobile
Ideas
Kvetch!
Search Plugins
Search
Popular Tags More »
widget (2087)
Post (1470)
plugin (1288)
posts (1116)
WordPress Simple Paypal Shopping Cart
Very easy to use Simple WordPress Paypal Shopping Cart Plugin. Great for selling products online in one click from your WordPress site.
Description Installation FAQ Screenshots Other Notes Changelog Stats
WordPress Simple Paypal Shopping Cart allows you to add an 'Add to Cart' button on any posts or pages. It also allows you to add/display the shopping cart on any post or page or sidebar easily. The shopping cart shows the user what they currently have in the cart and allows them to remove the items. It can be easily integrated with the NextGen Photo Gallery plugin too.
For screenshots, detailed documentation, support and updates, please visit: http://www.tipsandtricks-hq.com/?p=768 or http://www.tipsandtricks-hq.com/ecommerce/wp-shopping-cart
Author: Ruhul Amin
Tags: online shop, Paypal shopping cart, sell digital products, shopping cart, wordperss ecommerce, WordPress shopping cart
Download Version v2.8.4
Requires WordPress Version: 2.6 or higher
Compatible up to: 3.0.5
Last Updated: 2010-6-28
Downloads: 79,268
Average Rating
(70 ratings)
FYI
Other Versions »
Donate to this plugin »
Forums Posts

Nate Sanders from the PayPal team (@ppnsanders) wrote a detailed blog post on this plugin several months ago. Click here to read Nate's post, which includes screenshots and details on how to use the plugin once you have it installed.

Now let's try a different search. If you search for "PayPal" and "donation" and then look at the top few hits by popularity and rating, the "PayPal Donations" plugin stands out. According to its Plugin Directory page (shown below), this plugin adds a donation shortcode and sidebar widget to WP.

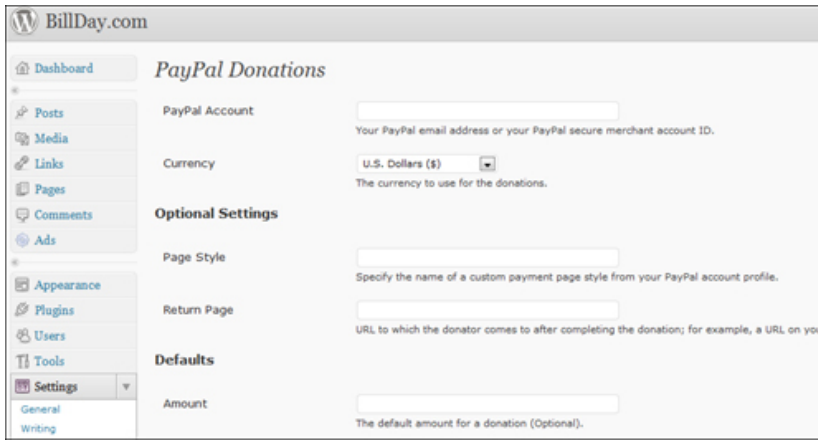
WordPress.org
Home Showcase Extend About Docs Blog Forums Hosting Download
Plugin Directory
Extend Home
Plugins
Developer Center
Themes
Mobile
Ideas
Kvetch!
Search Plugins
Search
Popular Tags More »
widget (2087)
Post (1470)
plugin (1288)
posts (1116)
PayPal Donations
Easy and simple setup and insertion of PayPal donate buttons with a shortcode or through a sidebar Widget. Donation purpose can be set for each button
Description Installation FAQ Screenshots Changelog Stats
Adds a PayPal donation shortcode and sidebar Widget to WordPress. The options menu lets you setup you PayPal ID and a few other optional settings. You can choose which donation button you want to use or if you want to use your own button. You can also set an optional default purpose and reference which can be overridden on each inserted instance with the shortcode options or in the Widget settings. There is also options available for currency, localization of the button, custom payment page style and the return page.
Widget
In the Appearance -> Widgets in WordPress 2.8 or newer you'll find the PayPal Donations widget. After adding it to your sidebar you can enter a title for the Widget, some descriptive text that will appear above the button and specify an optional purpose and reference for the donation button to override the default settings.
Download Version 1.4.9.3
Requires WordPress Version: 2.7 or higher
Compatible up to: 3.1.1
Last Updated: 2011-4-18
Downloads: 65,106
Average Rating
(27 ratings)
FYI
Other Versions »
Donate to this plugin »
Forums Posts

If you're interested in accepting donations, this plugin may be just what you need. In that case, you can either click the Download button in the Plugin Directory page, or you can download the plugin from within your WP installation. To do the latter, you would log in to the administration interface, click "Plugins" in the left sidebar, and then click "Add New" next to the Plugins page title, as shown in the next image.



Clicking "Add New" gives you a page where you can perform a Plugin Directory search. A search for "PayPal Donations" yields the result shown in the following figure.

From there you can click "Install Now." After the installation completes, you need to select "Activate Plugin" and then go to "PayPal Donations" under the "Settings" section to configure your installation, as shown in the next image.



Complete and save your configuration, and *voilà*—you're ready to start accepting PayPal-based donations!

If you're interested in seeing what other PayPal-based WP plugins are available, you should spend some time searching in the Plugin Directory to see what you can turn up. One search that currently returns no results is "PayPal" plus "embedded payments." This seems like useful functionality that's not currently available to WP users. I'll explore that and other missing functionality in the next article in this series.

Plugin Learnings and Where They'll Take Us

In this article, I've provided a quick overview to WordPress plugins, made some recommendations for useful plugins that can also give you a good idea of what's possible in WP, and discussed how to find PayPal-related plugins in the WordPress Plugin Directory. I've also dug a bit into two popular PayPal-related WP plugins, one for integrating a shopping cart and the other for accepting donations.

Next time we'll consider how to build a new plugin to provide currently missing PayPal functionality for WordPress users. Until then, please send in your questions and suggestions via the comment form on the article's page on x.com. Thanks in advance for your feedback!

About the Author

Bill is founder of Day Web Development, where he provides technical writing and editing, specification development, and platform and technology evangelism services. Recent projects include writing source code–level documentation, articles, partner how-to materials, conference session descriptions, and blog entries for O'Reilly Media, NVIDIA, Nokia, and Digital Reasoning Systems. In addition to being an active writer and blogger for the PayPal X Developer Community, Bill is also a contributor to GeekDad.com. Learn more and contact Bill via his LinkedIn profile and BillDay.com.

CHAPTER 23

Integrating Payments into WordPress, Part 3: Build Your Own PayPal Plugin

Bill Day

This is the third and final article in my series on using the WordPress weblog and CMS system.

In the first article, I introduced WordPress (aka WP), provided links to documentation to help you get up to speed with WP, and then showed you how to install and start using it. In the second installment, I provided an overview of WP plugins, made some general plugin recommendations, and showed you how to search for PayPal-related plugins in the WordPress Plugin Directory. I also demonstrated two popular PayPal plugins—one a shopping cart, the other a plugin to accept donations.

In this article, I'll detail the WordPress plugin model. We'll get our hands dirty with some code for an example WordPress plugin, and by the end, you'll know how to go about creating a simple WP plugin of your own.

A Note About PHP

To hack around in WordPress, and especially to modify or write new WP plugins, you will need to have some PHP programming knowledge. The good news is that PHP is a relatively straightforward language. There are also some very helpful resources for getting up to speed quickly with it.

If you just want a minimalist, WP-oriented introduction to PHP, I'd recommend you work through Adam Brown's "PHP Tutorial for WordPress Users." If you want to go further, dive into the official PHP manual and

tutorial. These resources should give you the prerequisite PHP knowledge to work through the rest of this article.

Understanding "The Loop"

There's one more major item you need to understand before we delve into WP plugin code. That item lies at the core of how WP renders post contents. It is The Loop.

The WP Codex summarizes The Loop like this:

> *The Loop is used by WordPress to display each of your posts. Using The Loop, WordPress processes each of the posts to be displayed on the current page and formats them according to how they match specified criteria within The Loop tags. Any HTML or PHP code placed in the Loop will be repeated on each post. When WordPress documentation states "This tag must be within The Loop", such as for specific Template Tag or plugins, the tag will be repeated for each post.*

Some plugins operate within The Loop, others outside of it. As you design a plugin, keep The Loop in mind and make sure that, if your plugin will be manipulating how post contents display, it's set up to be executed within the cycles of The Loop. If you have any questions about what The Loop is or how to interact with it, I would encourage you to take your time reading through its Codex page and the resources linked to from it, including "The Loop in Action."

Note that the "The" is always capitalized in "The Loop" because of its great importance in the WP system. (A bit high and mighty, I guess, but just go with it.)

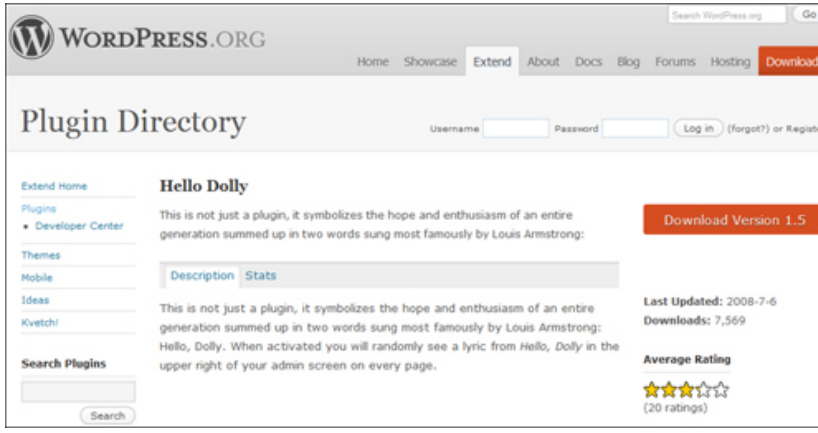
The Anatomy of a WordPress Plugin

It's time to start looking at plugin code. The Codex page "Writing a Plugin" provides a high-level overview of the steps required and where to go to learn more about each. In the first few steps, you'll create:

- A unique plugin name; search through the Plugin Directory to make certain your name is unique so that you can avoid naming collisions later.
- One or more plugin files; you must have at least one PHP file containing your plugin code, but you may have additional files (JavaScript, CSS, images, etc.).
- Optional: A `readme.txt` file to provide more information if you list your plugin in the Plugin Directory; the format for your readme is described here.

- Optional: A home page for your plugin if you are going to distribute it to others; this page should describe what the plugin does, how to install it, version compatibility and prerequisites, how to use the plugin, etc.—you may either use your Plugin Directory page or create a page on your own site for this.

Let's look at an example plugin—the first plugin ever written for WP, "Hello Dolly," created by the WP's founding developer. This is a very good plugin to examine because it's simple and it's included with WP installations by default. If you're following along with this article series and have installed WP, you should already have a copy of Hello Dolly on your server. If for some reason you don't see it listed on your server's plugin administration page (`http://yourserver/wp-admin/plugins.php`), you can install it from its Plugin Directory page, shown below.



Hello Dolly consists of only one PHP file, `hello.php`, contained within a `hello-dolly` directory in the downloadable ZIP distribution and shown here:

```
<?php
/*
Plugin Name: Hello Dolly
Plugin URI: http://wordpress.org/#
Description: This is not just a plugin, it symbolizes the hope and
 enthusiasm of an entire generation summed up in two words sung most
 famously by Louis Armstrong: Hello, Dolly. When activated you will
 randomly see a lyric from <cite>Hello, Dolly</cite> in the upper right
 of your admin screen on every page.
Author: Matt Mullenweg
Version: 1.5
Author URI: http://ma.tt/
*/

// These are the lyrics to Hello Dolly
```

```
$lyrics = "Hello, Dolly
Well, hello, Dolly
It's so nice to have you back where you belong
You're lookin' swell, Dolly
I can tell, Dolly
You're still glowin', you're still crowin'
You're still goin' strong
We feel the room swayin'
While the band's playin'
One of your old favourite songs from way back when
So, take her wrap, fellas
Find her an empty lap, fellas
Dolly'll never go away again
Hello, Dolly
Well, hello, Dolly
It's so nice to have you back where you belong
You're lookin' swell, Dolly
I can tell, Dolly
You're still glowin', you're still crowin'
You're still goin' strong
We feel the room swayin'
While the band's playin'
One of your old favourite songs from way back when
Golly, gee, fellas
Find her a vacant knee, fellas
Dolly'll never go away
Dolly'll never go away
Dolly'll never go away again";

// Here we split it into lines
$lyrics = explode("\n", $lyrics);
// And then randomly choose a line
$chosen = wptexturize( $lyrics[ mt_rand(0, count($lyrics) - 1) ] );

// This just echoes the chosen line, we'll position it later
function hello_dolly() {
        global $chosen;
        echo "<p id='dolly'>$chosen</p>";
}

// Now we set that function up to execute when the admin_footer
   action is called
add_action('admin_footer', 'hello_dolly');

// We need some CSS to position the paragraph
function dolly_css() {
        echo "
        <style type='text/css'>
        #dolly {
                position: absolute;
                top: 2.3em;
                margin: 0;
                padding: 0;
                right: 10px;
```

```
                font-size: 16px;
                color: #d54e21;
        }
        </style>
        ";
}

add_action('admin_head', 'dolly_css');

?>
```

At the top of the Hello Dolly source code listing, you see some standard plugin information. WP uses this header information to recognize the plugin, add it to the plugin administration screens, and allow you to activate/deactivate it on your server. The header format is detailed on "Writing a Plugin" as:

```
<?php
/*
Plugin Name: Name Of The Plugin
Plugin URI: http://URI_Of_Page_Describing_Plugin_and_Updates
Description: A brief description of the Plugin.
Version: The Plugin's Version Number, e.g.: 1.0
Author: Name Of The Plugin Author
Author URI: http://URI_Of_The_Plugin_Author
License: A "Slug" license name e.g. GPL2
*/
?>
```

Note that at least `Plugin Name` must be provided. Some plugins may omit one or more of the other header lines. Hello Dolly, for instance, does not provide a `License` line.

Next in the source you'll see the lyrics to the song "Hello, Dolly!" and some code that splits the lyrics up into lines based upon newline characters. The code then randomly chooses a line to display.

After the lyric logic, we get into the meat of the plugin, the `hello_dolly` and `dolly_css` functions. Action "hooks" set these two functions up to execute when certain actions occur within the WP system. I discussed the WP Plugin API's action and filter hooks in the previous article in this series; please refer to that article if you need a refresher on WP hooks.

For our purposes here, the key point is that `hello_dolly` hooks into WP via the `admin_footer` action (executed at the end of the administration panel), while `dolly_css` hooks in via `admin_head` (executed in the HTML `head` section of the admin interface). In other words, each function is registered as a callback such that when the specified administrative interface action occurs, the function is executed and the previously chosen line from the song is displayed in the upper-right portion of the WP admin interface. Wrap your brain around

this critical callback functionality, and WP plugin programming becomes fairly straightforward.

By the way, the same gentleman that provides the "PHP Tutorial for WordPress Users" discussed previously also maintains a very useful "WordPress Hooks Database." I highly recommend it for learning about any and all of the large number of action and filter hooks built into WP and available for your use.

PayPal Functionality in Our Plugin

Now things get more interesting. In brainstorming and asking X.com community members for ideas for what the PayPal functionality in my example plugin should do, I hit upon a simple but useful feature that I haven't seen in any other WP plugin: getting your current PayPal account balance.

PayPal provides some PHP code for getting one's balance. This code is available from its PayPal Sample Code page. You can access this `GetBalance` name-value pair (NVP) PHP example code under the "PayPal APIs" section of that page, or directly here.

I decided to use that example, along with what we've learned above from Hello Dolly and elsewhere, to put together my own WP PayPal GetBalance plugin. Like Hello Dolly, all of its logic is located in a single PHP file—in this case, in `billday_paypal_getbalance.php`. Note that I've prepended `billday_paypal_getbalance_` to all variable names to make certain they are unique in the WordPress systems so that my plugin will play nicely with others. I've also used function names beginning with `BillDay_PayPal_` for the same reason.

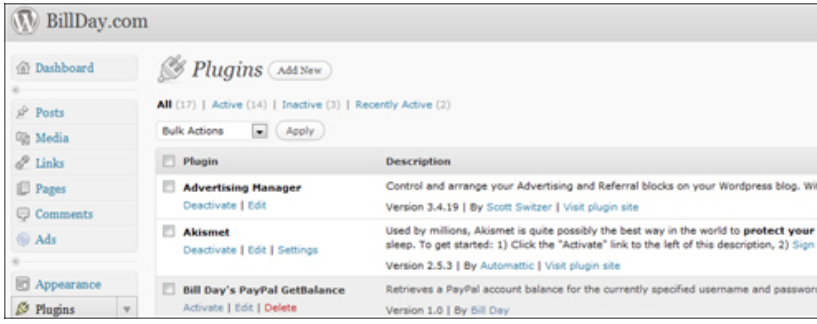
Deploying and Testing the Plugin

Once you've saved a copy of the plugin file, you can test-deploy it in your WP installation by doing the following:

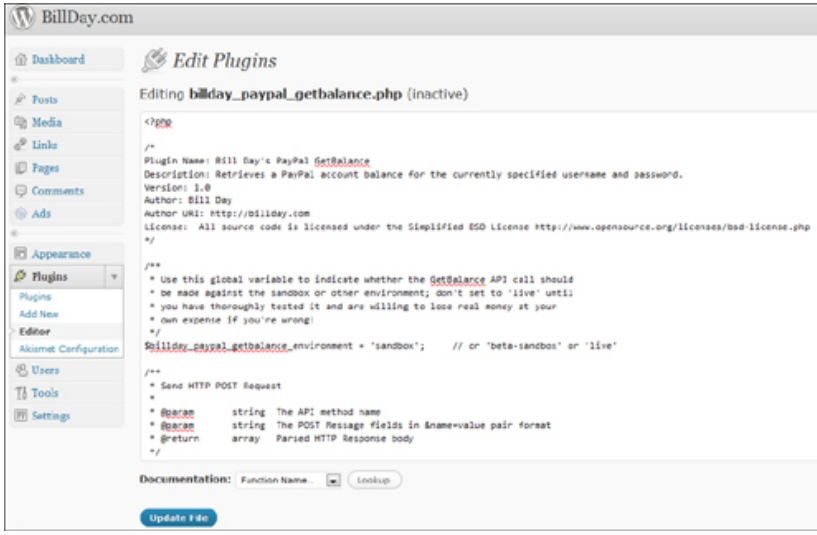
1. Save a local copy of `billday_paypal_getbalance.php` to your development system.
2. Edit your copy to change `my_api_username`, `my_api_password`, and `my_api_signature` values (for development, you should use your PayPal Sandbox test API user/pass/sig) and save the edited file; alternatively, you can edit these values after the following step, using the WP "Edit Plugins" admin screen if you prefer.
3. Copy `billday_paypal_getbalance.php` into the `wp-content/plugins` directory on your server.

4. Activate the new plugin by logging into your WP administration interface and selecting "Plugins."

Here's what the plugin looks like after you've copied it to your WP Plugin Directory but before you've activated it.


BillDay.com
Dashboard
Posts
Media
Links
Pages
Comments
Ads
Appearance
Plugins
Plugins
Add New
All (17) | Active (14) | Inactive (3) | Recently Active (2)
Bulk Actions
Apply
Plugin
Description
Advertising Manager
Deactivate | Edit
Control and arrange your Advertising and Referral blocks on your Wordpress blog. With
Version 3.4.19 | By Scott Switzer | Visit plugin site
Akismet
Deactivate | Edit | Settings
Used by millions, Akismet is quite possibly the best way in the world to protect your bl
sleep. To get started: 1) Click the "Activate" link to the left of this description, 2) Sign up
Version 2.5.3 | By Automattic | Visit plugin site
Bill Day's PayPal GetBalance
Activate | Edit | Delete
Retrieves a PayPal account balance for the currently specified username and password.
Version 1.0 | By Bill Day

Should you choose to edit the user/pass/sig information from within the WP interface, you would click on "Edit" underneath the not-yet-activated plugin. This brings up the screen where you can make edits for your particular account information, as shown here.



When you're ready, activate the plugin and enjoy!

Things to Learn from This WordPress Series

I hope you've learned through this series how to install and use the WordPress system (assuming you weren't using it already), how to add new features and capabilities with third-party plugins, and even how to build your own payments-related WP plugins using the PayPal APIs. I also hope you'll take this starting point and build some fantastic PayPal plugins! And if you do, please consider sharing them by contributing them back to the WP Plugin Directory. Good luck, and happy payments hacking!

About the Author

Bill is founder of Day Web Development, where he provides technical writing and editing, specification development, and platform and technology evangelism services. Recent projects include writing source code–level documentation, articles, partner how-to materials, conference session descriptions, and blog entries for O'Reilly Media, NVIDIA, Nokia, and Digital Reasoning Systems. In addition to being an active writer and blogger for the PayPal X Developer Community, Bill is also a contributor to GeekDad.com. Learn more and contact Bill via his LinkedIn profile and BillDay.com.

CHAPTER 24

Crowd Marketing

Trevor Cornwell

The core challenge behind most new ideas isn't funding—it's sales. Most companies think they know what is involved to get sales. But if getting sales were that quantifiable, any rational company would simply do the math and pay its way through to linear growth or beyond. You can calculate the cost to drive traffic—maybe. With some experimentation, you can figure out what ads cost and then measure how many users click through to your site. But then what? How many buy? How many come back later and then buy? And even if you can, and do, measure all of that, you are still left with the question of *why*. Why are users buying or not buying? Is it the UI? Your product? What most companies punt to is the hope that their product will go viral. To be sure, a lot of companies *do* go viral—and this concept extends beyond the YouTube video that gets a million views. Companies like Box.net or Dropbox attract large numbers of users by having a great product that people want. Friends tell other friends, and all of a sudden the cost of acquiring a customer drops to a very low number.

Viral Isn't a Plan—It Is a Consequence

When asked what their marketing plan is, many people answer "this can go viral." But viral is not a plan; it's a consequence of doing a lot of things well. The product has to be great—that is the first essential. The cost of switching is not so high that mediocrity can be tolerated. At a traditional retail store, good spreads slowly, and bad spreads quickly. Marketing can move faster than word of mouth. With an online, digital good, the reaction is lightning-fast. But that doesn't mean you can simply build something great and they will come. To market ideas, you need to harness your network and put it to work.

Making a Good Fire

The future is in *crowd marketing*. Marketing works in a series of tight, concentric circles. The people you know come to love your idea or product, and their network is connected to the next circle outward, which connects to the next circle, and so on. This is a powerful way to promote an idea or a concept; with crowd marketing, you can move from circle to circle.

The first step is organizing your fan base. Start with people you know, and make sure that they are aware of your idea. That is really the easy part, but it's one that is often ignored. The next step is figuring out the affinity audience for your idea. This requires the most imagination but also has the most potential to spread your idea like brush fire. Your friends and family are the match that lights the fire; the people who will have an interest in your idea are the kindling. To make sure your idea catches, you need to set up the fire just like you would in your fireplace. Better yet, think about setting the fire when you are camping, you have only a few matches, and it is damp outside. Getting everything set up well is critical, and you likely don't have a second chance. Your first circle of friends will help, but the spark or small flame that is created needs to have every chance to catch on. You have a 1,000 times better chance of succeeding if you get your fire set up right.

What Happened?

We tend to ignore these steps after we have spent a lot of time perfecting everything else: concept? check; engineering? check; price? check. Then we launch something, and what happens? Nothing. It may not have anything to do with a bad product. A lot of good ideas fail because they don't receive the right marketing.

Bands have fans. Movies have fans. Even products have fans. You have fans. It's time to put them to work.

OK, so where do you find all the fans if you don't have a publicist or a marketing firm? Many exist in the *keywords* of your idea. You can find crowds and fans in Facebook and Twitter. Facebook knows a lot about what engages its users just by virtue of what they are talking about on the site. You can mine for crowds by marketing to people you know on Facebook who are talking about subjects or using keywords related to your business. That is a powerful way to find the people who already like what you are doing. The same goes for Twitter: you can find what people are talking about related to your idea and reach them through a dialogue about your shared interest.

Life Ahead Is eBay Without Having to Ship Stock from the Attic

But what is the future? eBay paved a new path by creating peer-to-peer selling for the things we have in our attics. Brilliant. Match a product you have with a need that someone has, don't limit the sellers to just stores, and build trust through reputation. That model was destined for success because it understood a few things: there are a thousand ways to market, selling isn't something for just big stores, and trust can be scored. What now seems obvious was, a few years back, revolutionary. So does this strategy apply to digital goods? You bet. Distribution is a huge pain point for developers and apps. Apple and Android have created consignment stores, but getting attention for a new app isn't easy. How can you do it? First, stick with basic principles. Identify people you know and ask them to buy your product. Engage in a dialogue by talking to people on Twitter. Create a fan page to share and report and excite people on Facebook. Your blog is a chance to invite people behind the scenes to see what you are working on. We all have enough professional marketing hitting us every day. Take a moment to talk about the struggles that you have had and share the learning points; people will come back again and again to identify and connect with you—and your product. eBay's CEO calls the company the first social network. That may be right. It is social to sell, but just like with physical goods, you don't market tires to people who can't drive. Find your specific audience and engage it.

Let Them Buy Wholesale and Distribute

appbackr takes the approach to allow fans to buy wholesale and sell retail, and was inspired by the eBay model. A key to that "first social network" is enabling people to promote their products in their own way. Another key element is allowing the sellers to profit from the difference in the value that they give to the product and the value that a retail customer is willing to pay. The market for digital goods allows for the same model. Sometimes it is assumed that for a digital good—since there is no incremental cost for manufacturing it or costs for shipping it—that distribution is not necessary. But in a digital world, distribution is social. Note, though, that just because it is social doesn't mean that your "friends" will do it for free. At appbackr, we allow "backers" to buy digital apps at a wholesale price and profit when the apps sell at the retail market. What is the value of the backer? Well, he can buy in bulk so that the developer gets immediate payment for thousands of apps at once. That has a value because it benefits cash flow, allows for the creation of new features, and creates a marketing budget for new apps. But the real benefit comes in the

crowd marketing for apps. Imagine a thousand backers each working to promote an app. That can have a huge effect in helping the app to achieve top-10 status, which in turn helps it to sell more.

Fans, Friends, Launchpads, and New Markets

The idea is simple: make each fan a marketer. If you view each of your fans as having her own launchpad to a thousand different friends, colleagues, and followers, you begin to see the promise of crowd marketing. That is the future of the Internet—and the social infrastructure that Facebook and Twitter and their hundreds of millions of users have built. Your product has fans, and those fans are connected to others. The social graph is your way of connecting your product to the people who want it.

About the Author

Trevor Cornwell is the founder and CEO of appbackr inc, the first wholesale marketplace for apps. appbackr was a winner of the PayPalX 2010 Developer Challenge.

CHAPTER 25

Bitcoin: Money Without Governments

Bill Day

I recently completed a six-part series examining alternative payment systems and comparing them to the PayPal X Platform. As I wrapped that up, I asked readers to weigh in on which topic they'd like to read about next. I received votes for several topics on the list, including a request for coverage of the eBay APIs (watch for more from me on that topic coming soon). But the one request that caught me off guard was for an article on Bitcoin.

Bitcoin is a relatively new digital currency. But there's much more to it when you dig a bit under the covers. Shall we?

What Is Bitcoin?

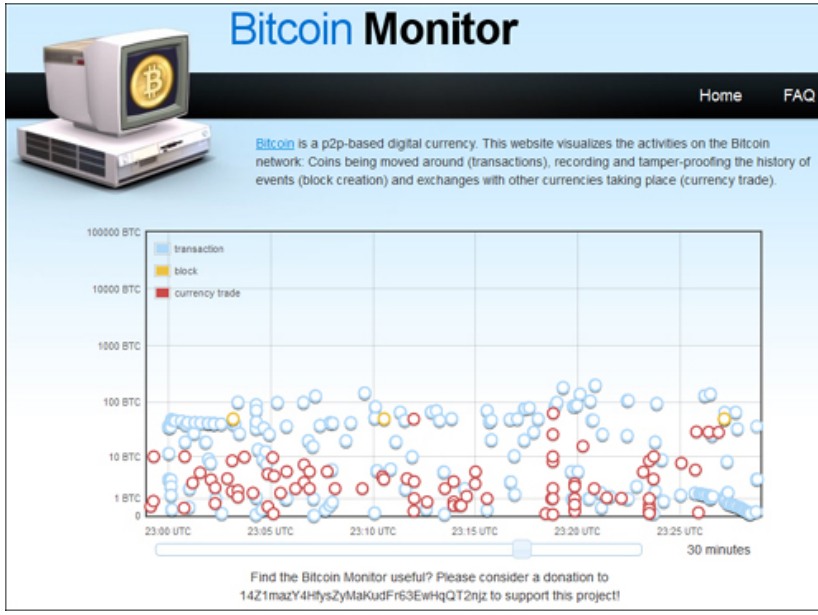
Here's the high-level description of what Bitcoin is and provides, taken from its Wikipedia page:

> Bitcoin is a digital currency created in 2009, based mainly on a self-published paper by Satoshi Nakamoto. [It] enables rapid payments (and micropayments) at very low cost, and avoids the need for central authorities and issuers. Digitally signed transactions, with one node signing over some amount of the currency to another node, are broadcast to all nodes in a peer-to-peer network. A proof-of-work system is used as measurement against double-spending and initial currency distribution mechanism.

The Bitcoin site WeUseCoins.com provides a video overview of what Bitcoin is, how the coins are created, and what you can buy with them at *http://www.youtube.com/watch?v=Um63OQz3bjo*.

For more detail, you can also watch a short Ignite presentation from the current Principal of the Bitcoin project, Gavin Andresen (@gavinandresen). Go to *http://www.youtube.com/watch?v=koIq58UoNfE&t=30* to watch Gavin's presentation on YouTube.

You can get a feel for the frequency of Bitcoin transactions by watching them occur via the Bitcoin Monitor site, shown below.



Interested in the nitty-gritty of how the system works? You can dig deeper by reading Nakamoto's original paper, "Bitcoin: A Peer-to-Peer Electronic Cash System" (click here for PDF). There is also a Bitcoin wiki with links to much more information, a FAQ, discussion forums, and developer information. You can also link to more information on the Bitcoin project and community via Bitcoin.org.

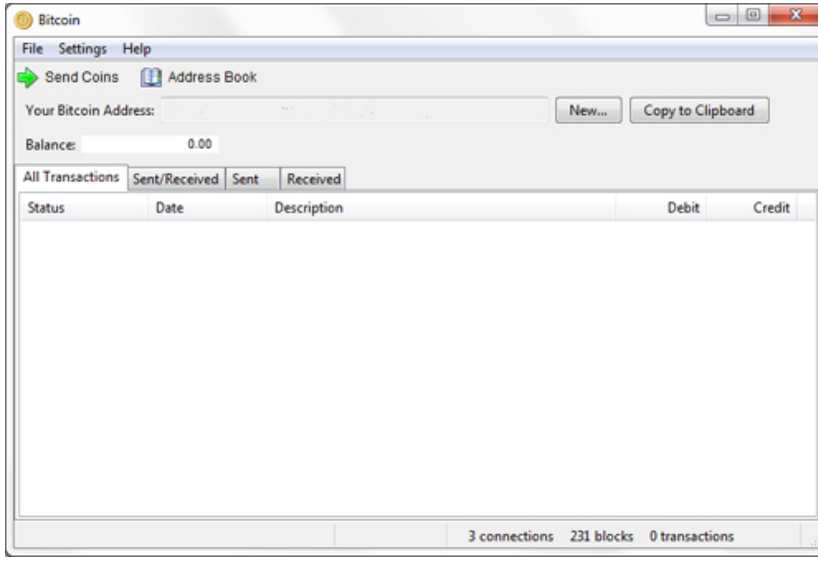
One very important thing to note: The Bitcoin system is based upon public-key cryptography. Without the crypto, you'd have no Bitcoin. Bitcoin being crypto-currency leads to some interesting properties, not the least of which is that it cannot be inflated by a central bank or governmental authority. This means that Bitcoins have commodity-like properties similar to gold and other precious metals.

Bitcoin's being based upon public-key cryptography also means that there are concerns about it being used for nefarious purposes. Governments and organizations interested in cryptography, money laundering, and cyberattacks are taking close looks at Bitcoin. We'll discuss these issues more later.

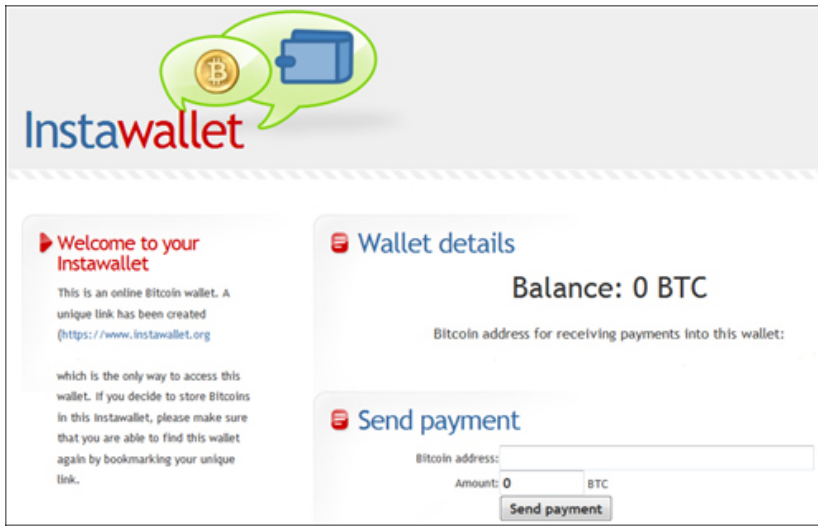
Acquiring and Storing Bitcoins

To use Bitcoins, you must first have a Bitcoin wallet. You can either download a piece of wallet software to install on your computer (click here for links to download the official Windows, Mac, or Linux client, or here to grab the source from GitHub) or use an online wallet. There are options for the latter listed on the "Getting Started with Bitcoin" page from WeUseCoins.

The official Windows wallet starts up like this (note that the provided Bitcoin Address has been removed from this screenshot).



Here's an example online wallet, Instawallet, which creates a new wallet for you by default when you visit its home page.


Instawallet
Welcome to your Instawallet
This is an online Bitcoin wallet. A unique link has been created (https://www.instawallet.org
which is the only way to access this wallet. If you decide to store Bitcoins in this Instawallet, please make sure that you are able to find this wallet again by bookmarking your unique link.
Wallet details
Balance: 0 BTC
Bitcoin address for receiving payments into this wallet:
Send payment
Bitcoin address:
Amount: 0
BTC
Send payment

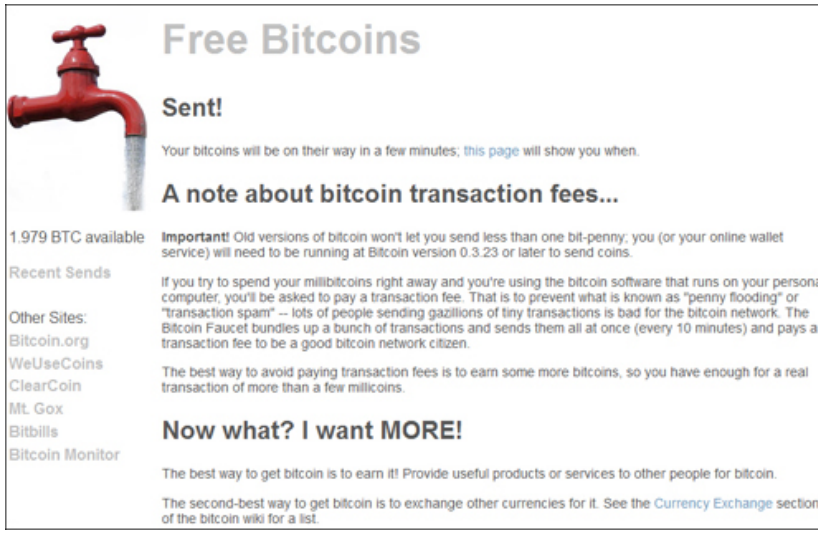
To continue using the same Instawallet in the future, you need to bookmark the unique wallet URL assigned to you. The Instawallet site notes that you should only view this option as a "spare change" wallet, rather than storing a large amount of Bitcoinage here.

Whether you choose to use a downloaded wallet or an online one, there are some potential gotchas to consider. If you lose your downloaded wallet—say through a system crash or theft—you also lose any coins stored in it. Likewise, if you use an online wallet such as Instawallet and lose the URL or authentication information for that wallet, or the service provider disappears, or anything else keeps you from reaching your online wallet, you may also lose coins. So be sure you have a backup and security strategy in place to avoid any problems later.

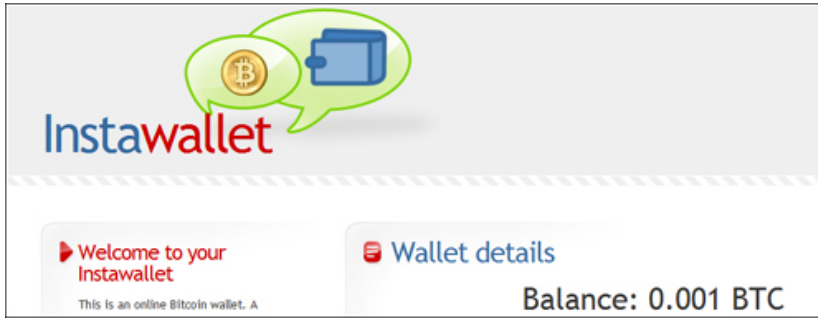
Whichever approach you choose, once your wallet is ready, you will have a new Bitcoin Address. You use this address to send and receive coins.

So how can you get some Bitcoins to try out the system? There are several options listed on the aforementioned "Getting Started with Bitcoin" page. But perhaps the easiest option is to use a free service from Gavin Andresen, Free Bitcoins. Gavin's service will transfer a small amount of coinage (0.001 Bitcoin, or BTC, at the time of this writing) for free to whatever receiving Bitcoin Address you specify. The only catch is that you need a Google account (free to acquire if you don't already have one).

For example, I entered my Google account information and Bitcoin Address for Instawallet into the Free Bitcoins "faucet," and it sent 0.001 BTC my way, as shown here.


Free Bitcoins
Sent!
Your bitcoins will be on their way in a few minutes; this page will show you when.
A note about bitcoin transaction fees...
1.979 BTC available
Recent Sends
Other Sites:
Bitcoin.org
WeUseCoins
ClearCoin
Mt. Gox
Bitbills
Bitcoin Monitor
Important! Old versions of bitcoin won't let you send less than one bit-penny; you (or your online wallet service) will need to be running at Bitcoin version 0.3.23 or later to send coins.
If you try to spend your millibitcoins right away and you're using the bitcoin software that runs on your personal computer, you'll be asked to pay a transaction fee. That is to prevent what is known as "penny flooding" or "transaction spam" -- lots of people sending gazillions of tiny transactions is bad for the bitcoin network. The Bitcoin Faucet bundles up a bunch of transactions and sends them all at once (every 10 minutes) and pays a transaction fee to be a good bitcoin network citizen.
The best way to avoid paying transaction fees is to earn some more bitcoins, so you have enough for a real transaction of more than a few millicoins.
Now what? I want MORE!
The best way to get bitcoin is to earn it! Provide useful products or services to other people for bitcoin.
The second-best way to get bitcoin is to exchange other currencies for it. See the Currency Exchange section of the bitcoin wiki for a list.

After a little bit of time for the network communication and computations to happen in the background, the Bitcoinage appeared in my Instawallet.

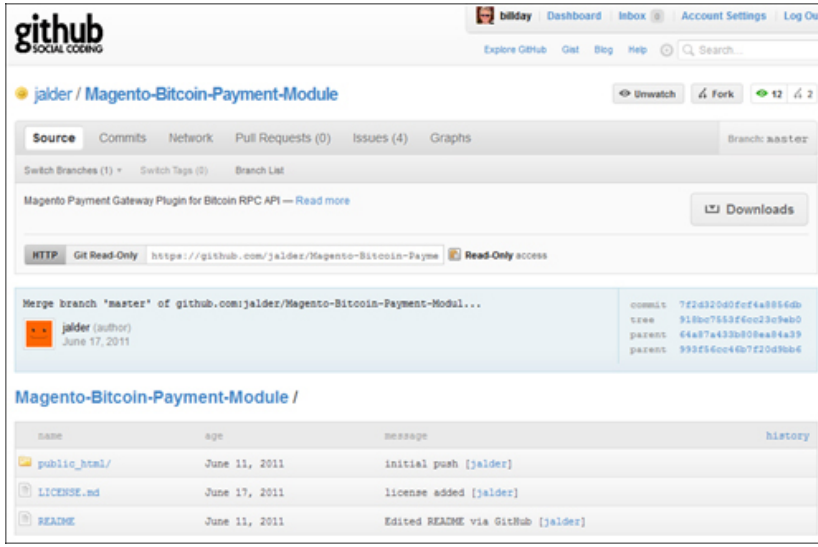


There are a number of other ways you can add coins to your wallet. You can earn them, purchase them in a market (check out Bitcoin Charts for a view into the various markets and currency-to-Bitcoin purchases), and even buy them over the counter (OTC). Bitcoins can also be created via a process known as *mining*, but for most consumers and merchants interested in Bitcoins for commerce but not in becoming a mining business, the amount of computational horsepower required may be prohibitive (see this ZDNet article for more on mining).

Using and Accepting Bitcoins

However you acquired your Bitcoins, once you have some, you will probably want to start using them. There is a wide and growing variety of legitimate services and merchandise being sold today for Bitcoins. Visit the Trade page on the Bitcoin wiki to see a list of many of the available items.

What if you want to accept Bitcoins as a method of payment? The same wiki contains a Merchant Howto to help you get started. It provides information on manually accepting Bitcoins as well as automating Bitcoin-based purchases. And you can automate purchases via existing shopping cart solutions (for example, a Magento Payment Gateway Plugin, shown below) or programmatically yourself using a JSON-RPC interface.



For more information on the Bitcoin ecosystem, I would recommend reading through the linked resources on Dave Mackey's "A Bitcoin Primer."

Threats and Opportunities

As Bitcoin has gained somewhat wider acceptance and much more visibility, it's received a lot of criticism. There have been exchange data issues, threats against the system because of its use in black-market purchases for illegal items such as drugs, and claimed heists of large sums of Bitcoinage. There has also been some incredible speculation that saw the value of 1 BTC, as expressed in US dollars, climb nearly 1000% in a single month earlier this year! Things have

gotten so heated in this space that even the CIA is studying up on Bitcoin. So what are the significant threats and opportunities for it?

One major threat to Bitcoin's long-term success is ease of use. Needing to have a working knowledge of "crypto-currency" and "peer-to-peer" may not turn off techies, but it certainly confuses a lot of other people. If people can't understand Bitcoin, they won't use it. If not many people use Bitcoin, it will continue to see non-sticky value jumps versus other currencies. And large swings in value lead to a further lack of confidence in any currency or commodity.

The opportunity inherent in this problem lies in making the Bitcoin system very easy to use. Implementers need to concentrate on bulletproof clients and merchant software if they hope to maximize Bitcoin uptake. And for goodness' sake, hide the cryptography details and never use the acronym "p2p" in Getting Started guides and consumer-oriented documentation!

Another threat to Bitcoin's viability is that it has in fact been used in certain high-profile illicit markets. The opportunity here is for legitimate Bitcoin users and supporters to educate their governmental representatives on the many valid uses of Bitcoin. Just as with dollars or any other currency or commodity, Bitcoin is inherently neither good nor evil. But unlike the other, more established value stores and means of commerce, Bitcoin *is* terribly new, and therefore poorly understood, which in some cases leads to fear. Education is the only way out of that.

Perhaps the biggest impediment to Bitcoin's long-term viability: lack of perceived need, at least in much of the developed world. As a recent TechCrunch article put it:

> Why would any American or European business adopt Bitcoin in the first place? It's an elegant and disruptive solution desperately looking for a problem.

As the article's author noted, the opportunity may lie in developing countries currently suffering hyperinflation. Bitcoin, or a similar inherently anti-inflationary currency, might be exactly what those countries need to help stabilize their monetary situation. Just as we've seen Kenyans jump over traditional banking systems to adopt M-PESA mobile phone–based banking, might we see some developing countries looking to jettison their tremendously inflated currency jump over today's well-established reserve currencies and instead choose to use a crypto-currency, Bitcoin or otherwise?

I'd love to read your take on Bitcoin and the opportunities ahead of it. Please leave a comment below with your thoughts.

About the Author

Bill is the Principal of Day Web Development, where he provides technical writing and editing, specification development, and platform and technology evangelism services. Recent projects include writing source code–level documentation, articles, partner how-to materials, conference session descriptions, and blog entries for O'Reilly Media, NVIDIA, Nokia, and Digital Reasoning Systems. In addition to being an active writer and blogger for the PayPal X Developer Community, Bill is also a contributor to GeekDad.com. Learn more and contact Bill via his LinkedIn profile and BillDay.com.